GUIA DO PROGRAMADOR

PROGRAMAÇÃO BÁSICA E AVANÇADA

TOSHIYUKI SATO
PAUL MAPSTONE
ISABELLA MURIEL

MSX
Guia do Programador
Programação Básica e Avançada
Volume 1

MSX
Guia do Programador
Programação Básica e Avançada
Volume 1

TOSHIYUKI SATO PAUL MAPSTONE ISABELLA MURIEL

Tradução:
Flávio Denny Steffen
Lars Gustav Erik Unonius

Revisão Técnica:
Oscar Burd
Daniel Burd

McGraw-Hill
São Paulo
Rua Tabapuã, 1.105, Itaim-Bibi
CEP 04533
(011) 881-8604 e (011) 881-8528

Rio de Janeiro • Lisboa • Porto • Bogotá • Buenos Aires • Guatemala • Madrid • México • New York • Panamá • San Juan • Santiago

Auckland • Hamburg • Kuala Lumpur • London • Milan • Montreal • New Delhi • Paris • Singapore • Sydney • Tokyo • Toronto

Do original
The Complete MSX – Programmers Guide

EDITOR: MILTON MIRA DE ASSUMPÇÃO FILHO

Coordenadora de revisão: Daisy Pereira Daniel
Supervisor de produção: José Rodrigues
Capa: layout: Cyro Giordano

Dados de Catalogação na Publicação (CIP) Internacional
(Câmara Brasileira do Livro, SP, Brasil)

Sato, Toshiyuki.
S266g MSX Guia do programador : Programação Básica e Avançada / Toshiyuki
v.1 e 2 Sato, Paul Mapstone, Isabella Muriel ; tradução Flávio Denny Steffen, Lars Gustav Erik Unonius. – São Paulo : McGraw-Hill, 1988.

1. Microcomputadores – Programação 2. MSX (Computadores) – Programação 3. MSX-BASIC (Linguagem de programação para computadores) I. Mapstone, Paul. II. Muriel, Isabella. III. Título.

CDD-001.6424
88-0192 -001.642

Índices para catálogo sistemático:

1. Microcomputadores : Programação : Processamento de dados 001.642
2. MSX : Computadores : Programação : Processamento de dados 001.642
3. MSX-BASIC : Linguagem de programação : Computadores : Processamento de dados 001.6424

SUMÁRIO

Parte I

PROGRAMAÇÃO

CAPÍTULO 1

ORGANIZAÇÃO

Quando você desempacotou o seu computador MSX, deve ter encontrado um cabo para conectar o vídeo, ou a TV, ao computador.

Por enquanto você só precisa conectar o seu computador à televisão. Numa TV colorida, você pode usar os comandos de cores do MSX, com mais vantagens mas, uma TV em branco e preto também serve.

Para conectar a TV ao computador, insira um dos conectores no soquete marcado com "TV" na parte traseira do computador; a outra ponta deve ser conectada à tomada da antena da TV. Se o seu televisor tiver dois soquetes, marcados com UHF e VHF, use o soquete UHF.

Ligue a TV e espere ela esquentar. Então ligue o seu computador e, se ele estiver corretamente sintonizado, você verá em primeiro lugar uma tela semelhante a esta:

---

sistema MSX
versão 1.0
Copyright 1983 by Microsoft

... e então, após uma pausa, aparecerá algo mais ou menos assim:

```
BASIC MSX versão 1.0
Copyright 1983 by Microsoft
28815 Bytes livres
OK




color  auto  goto  list  run
```

Se você não vir essa segunda tela, ajuste o sintonizador do seu TV até que consiga. (O número de bytes livres irá depender do tamanho da memória do seu computador.) Isso acontece sempre que o MSX é ligado.

Se o seu televisor tiver um controle de sintonização deslizante, ajuste-o até obter uma imagem clara. Se você tiver um botão de pressão para cada canal, é aconselhável sintonizar em um canal fora de uso.

Agora você está pronto para usar o teclado. Experimente da maneira que quiser, pois não há risco de danificar o seu computador! Tudo o que digitar aparecerá na tela da TV. Provavelmente você receberá muitas mensagens de erro do computador, mas não se preocupe, pois aprenderá como evitar isso mais tarde.

Se olhar para a tela, logo à primeira vista, irá ver um pequeno quadrado branco abaixo da mensagem "OK". Ele é chamado cursor, e mostra onde o próximo caractere irá aparecer na tela. Tente digitar algumas letras com ele. É bom adquirir o hábito de usar ambas as mãos enquanto estiver digitando. Pressionando qualquer tecla você faz com que apareça o caractere correspondente na tela. Digite:

```
A veloz raposa salta sobre o cachorro preguiçoso.
```

Para obter espaços, pressione a barra comprida na parte inferior do teclado. Para conseguir caracteres maiúsculos basta pressionar a tecla <SHIFT> e a tecla do caractere que você quer, simultaneamente.

No lado esquerdo do teclado existe uma tecla marcada com <CAPS LOCK>. Ao pressionar essa tecla, tudo o que você digitar depois aparecerá em letras maiúsculas. Para voltar à modalidade minúsculas, pressione essa tecla outra vez.

Agora olhe para os números. Veja o que acontece se você pressionar <SHIFT> e, ao mesmo tempo, as teclas com os números de 1 a 9.

Na parte superior do teclado existem cinco teclas alongadas marcadas com os caracteres F1 a F5. Essas são as teclas de função. A linha exibida na base da tela é destinada a essas teclas. Esqueça-as por enquanto, pois iremos ver o seu uso no Capítulo 8.

Agora encontre a tecla <STOP>. Você irá usá-la muitas vezes junto com a tecla Control, marcada com <CTRL>. A tecla <ESC> nada faz por enquanto, mas será usada para controlar a impressora.

A tecla <TAB> move o cursor ao longo da tela. Ela apaga qualquer caractere em seu caminho. Se o cursor estava originalmente no lado esquerdo, ele é movido para nona posição. Pressionando-se <TAB> outra vez, ele se movimenta para a décima sétima posição e, então, para a vigésima sexta posição.

A tecla <GRAPH> permite que você imprima caracteres gráficos, que são acessados pelas teclas de caracteres. Pressionando-se <L GRA> (ou <GRAPH> no HOT-BIT) (é L GRA ou R GRA no Expert) com qualquer tecla de caractere, obtêm-se caracteres gráficos. Você poderá ter diferentes caracteres gráficos se pressionar <SHIFT> <L GRA> e uma das teclas de caracteres ao mesmo tempo. Encontre os caracteres ♥, ♦, ♣ e ♠.

Pressionando <RGRA> (ou <CODE> no HOT-BIT) (LGRA ou RGRA no Expert) e uma tecla de caractere, você pode obter caracteres europeus ou gregos (por exemplo, α, β, ä, ü etc.) Experimente com essas teclas. Tente pressiona <SHIFT> também. A maioria das teclas fornecerá a você diferentes caracteres gráficos.

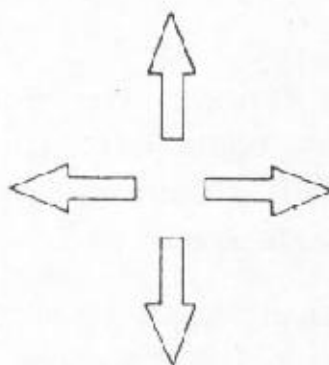
Uma tecla muito importante é a tecla <RETURN>. Sempre que quiser que o MSX leia o que você escreveu na tela, pressione <RETURN>. Se você pressioná-la agora, provavelmente obterá uma mensagem de erro. Não se preocupe, isso significa que o computador não entendeu o que você digitou.

Agora a tela deve estar toda cheia. Observe que quando chega ao fim da tela todo o texto se move para cima uma linha automaticamente e você perde a linha superior. Isso é chamado "rolamento", ou scroll.

Para limpar a tela, pressione <SHIFT> e <HOME> simultaneamente. <HOME> é uma das quatro teclas de edição na parte superior direita do teclado. Observe que após ter feito isso, o cursor salta para o canto superior esquerdo da tela. Sempre que a tela se tornar confusa, você pode limpá-la dessa forma. Pressionando-se essa tecla sem

pressionar <SHIFT>, o cursor volta para a parte superior esquerda da tela sem apagar o texto.

Agora digite uma linha inteira de caracteres. Para corrigi-la você deve pressionar a tecla de retrocesso <BS>. Isso cancelará qualquer coisa à esquerda do cursor. Da mesma forma que qualquer outra tecla, ela é repetitiva; se você mantê-la pressionada, irá, certamente, apagar a linha inteira.

Digite mais alguns caracteres e experimente pressionar as teclas do cursor:

Elas movem o cursor na direção marcada sem apagar o que encontram pela frente.

Tente apagar um caractere em particular usando as teclas <BS> e do cursor. Quando achar que captou a idéia, veja o que acontece quando você pressiona <DEL>, que é outra das quatro teclas de edição no canto superior direito do teclado.

Essa tecla apaga o caractere sob o cursor e move tudo à direita do cursor um espaço para a esquerda. Se você mantiver o dedo na tecla <DEL>, deletará tudo o que estiver à direita do cursor.

Suponha que você digitou o seguinte:

```
abcfg
```

Você pode querer inserir um "d" entre o "c" e o "f". Para fazer isso, mova o cursor até que o mesmo esteja sobre a letra "f" e então pressione a tecla de inserção <INS>. O cursor será substituído por uma linha em branco. Se você digitar "d", ele aparecerá inserido entre o "e" e o "f". Para fazer o cursor voltar ao modo normal, pressione <INS> outra vez. Pratique correções usando o cursor e as teclas de edição.

A quarta tecla de edição <SELECT> não faz nada por enquanto.

EXERCÍCIOS

1. Digite os dois caracteres gráficos associados a cada tecla, e o próprio caractere. Agora cancele os caracteres alfanuméricos, deixando somente os caracteres gráficos na tela.

2. Imprima o número 0 e a letra O; observe a diferença entre os dois caracteres. Você deve usar o 0 quando estiver digitando um número, pois de outra forma o computador informará um erro. Compare também o 1 com a letra minúscula l. Mais uma vez, não os confunda.

MODO DIRETO

Uma vez familiarizado com o teclado e com o uso das teclas <BS> e <INS>, você está pronto para digitar seus primeiros comandos no computador. Em primeiro lugar, limpe a tela pressionando as teclas <SHIFT> e <HOME> simultaneamente. Faça isso sempre que a tela estiver muito cheia ou confusa.

Agora digite cuidadosamente:

```
PRINT "Bem-vindo ao MSX"
```

e então pressione a tecla <RETURN>.

Você deve ver "Bem-vindo ao MSX" exibido na linha diretamente abaixo daquela que você digitou, com a palavra OK sob ela. Talvez, em vez disso, você receba uma mensagem de erro. Não se preocupe, repita o comando, pois provavelmente você cometeu algum erro de digitação.

Ao se pressionar <RETURN>, o computador executa um comando. Ele somente reconhece certas palavras, chamadas comandos, das quais PRINT é uma delas. Ao ver PRINT, seguido por alguma palavra entre aspas, ele sabe que é para imprimir tudo o que estiver incluído entre as aspas.

O computador não diferencia entre letras maiúsculas e minúsculas nos comandos e ignora espaços. Portanto, não adianta colocar espaços entre as letras de um comando, pois os mesmos não serão aceitos pelo computador.

```
print "Bem-vindo ao MSX"  <RETURN>
```

irá produzir o mesmo resultado.

Tudo o que você digitou entre as aspas será impresso exatamente como você digitou.

Para imprimir um número você não precisa de aspas. Digite:

```
PRINT 12345 <RETURN>
```

Você irá ver 12345 exibido diretamente abaixo da última linha impressa na tela. Da mesma forma, não importa quantos espaços você tenha digitado entre o comando PRINT e o número. Os espaços entre os números a serem impressos serão ignorados.

```
PRINT 12 345 <RETURN>
```

produz o mesmo resultado.

Agora digite:

```
PRINT 3+4 <RETURN>
```

o computador imprimirá o valor 7. Ele calculou a expressão 3+4 e forneceu o resultado. Se desejar imprimir o texto 3+4 como está, você deve digitar o seguinte:

```
PRINT "3+4" <RETURN>
```

O computador agora vê 3+4 como uma string, e a imprime exatamente como você a digitou.

Para subtrair um número, use o sinal "–".

Para multiplicar números, é usado o sinal "*", em vez do sinal de multiplicação mais comum "×". Isso foi feito para evitar a confusão com a letra "x".

Para dividir, use o sinal "/", em vez de ":".

Digite o seguinte, mas não se esqueça de pressionar <RETURN> após cada linha.

```
PRINT 4*3
PRINT "3+4 =";3+4
```

No último exemplo, você verá:

```
3+4 = 7
```

Quando o computador vê o ";", ele pensa que existe outro item a ser impresso, neste caso, a expressão 3+4. Dessa forma, ele calcula a expressão e a imprime logo após a string.

Observe agora que existe sempre um espaço em branco antes e após um número positivo.

```
PRINT "aa" ; 3;4;–5 "bb""cc"
```

irá fornecer:

```
aa 3  4 –5 bbcc
```

Como pode notar não são deixados espaços quando você imprime strings. Ao imprimir um número negativo, o sinal negativo é colocado no lugar do espaço em branco. Quando não é usada pontuação entre strings, o computador assume que foi colocado ponto e vírgula. Você deve usar pontuação entre os números, pois de outra forma o computador irá vê-lo como um número grande.

O computador também reconhece a vírgula em um comando PRINT, caso em que os itens serão impressos na mesma linha, sendo que o segundo será impresso 15 posições de caractere à direita do primeiro item impresso.

Agora digite as duas linhas seguintes:

```
LET A=3
PRINT A
```

O número 3 é exibido na tela.

O comando LET atribui um valor à variável, que neste caso (detectar) chamada A. No exemplo anterior, quando o computador recebe o comando PRINT, ele está esperando um número, pois não existem aspas. Ele então verifica em sua memória e descobre o valor que foi associado à variável A e, assim, imprime esse número.

Você pode ter expressões mais complicadas no comando LET:

```
LET B=31*72*4
PRINT B
```

O número 8928 é exibido.

Agora digite:

```
LET B=B+1000
```

Você obtém:

```
9928
```

O comando LET anterior parece matematicamente incorreto, mas ele é perfeitamente aceitável em computação. A expressão à direita de um sinal "=" é calculada e colocada na variável do lado esquerdo. Neste caso, a variável B é, agora, igual ao seu valor antigo mais 1000.

Nomes de variáveis devem começar com uma letra. Após essa letra podem ser usados números. Espaços em branco são ignorados em nomes de variáveis como:

A A é equivalente a AA

Podem ser usadas letras maiúsculas ou minúsculas, mas o computador não fará distinção entre as duas.

CARRO é equivalente a carro

O computador só lê os dois primeiros caracteres de um nome de variável, de modo que:

AB é equivalente a carro

É importante não usar nomes de comandos em um nome de uma variável, de modo que:

```
INTEIRO
```

é inválido pois INT é um comando.

Algumas palavras são reservadas para versões futuras da linguagem e não podem ser usadas dentro de nomes de variáveis, por exemplo:

```
NAME
```

é inválido, pois "NAME" é uma palavra reservada.

Eis alguns nomes de variáveis permitidos:

```
NÚMERO2
n2
NUM
```

Estes nomes não serão permitidos:

2n	O primeiro caractere deve ser uma letra.
num?	"?" é um caractere inválido em um nome de variável.

As strings também podem ser armazenadas como variáveis:

```
LET A$="Alô"
PRINT A$
```

É impresso Alô, já que PRINT A$ é o equivalente a digitar PRINT "Alô" nesse exemplo.

Nomes de variáveis string devem terminar com o sinal "$" para distingui-las de variáveis numéricas.

Eis alguns nomes válidos para variáveis string:

```
ANO$
RECEITA$
L1$
```

Estes nomes são inválidos:

A	Sem o sinal $.
A.B$	"." é um caractere inválido.
1PRIM$	O primeiro caractere deve ser uma letra.
ON$	ON é um comando.

TONAL$	ON é um comando.
NAME$	NAME é uma palavra reservada.

Outro comando útil é o INPUT. Digite:

```
INPUT A
```

Quando o computador percebe esse comando, ele exibe um ponto de interrogação e espera que você digite um número. Esse número é então associado à variável A. Digite um número. Você irá vê-lo aparecer após o ponto de interrogação. Se você digitar agora:

```
PRINT A
```

o número que você havia digitado será impresso. Se tivesse digitado uma string você obteria um erro.

EXERCÍCIO

1. Encontre o valor de A, onde:

```
A=49*38.07/13
```

CAPÍTULO 3

ESCREVENDO UM PROGRAMA

Você já trabalhou bastante no modo direto. Você digitou um comando na tela, pressionou <RETURN> e o computador executou o comando em seguida. Se quiser repetir o comando, você tem de digitá-lo outra vez, o que consome muito tempo.

Se quiser que o computador se lembre de um comando, você deve fornecer a ele um número de linha, por exemplo:

```
10 PRINT "ISTO E UM TESTE" <RETURN>
```

Ao pressionar <RETURN> o computador vê o número da linha, neste caso 10, e ele sabe que a linha é parte de um programa. Dessa forma, ele armazenará a linha inteira na memória. Ao terminar isso, ele exibirá o lembrete "OK" na tela.

Para fazer com que o computador execute a linha, o comando RUN deve ser usado. Digite:

```
RUN <RETURN>
```

e na tela aparecerá:

```
ISTO É UM TESTE
```

Para escrever um programa, cada comando deve ser precedido por um número de linha. Pode-se usar qualquer valor até 65529, mas é aconselhável iniciar com o 10 e

incrementar em valores de **10**, isto é, 10, 20, 30, 40 etc. Dessa forma, quando você tiver terminado de digitar o seu programa existirá espaço para inserir novas linhas entre as já existentes, bastando usar os números de linhas 15, 25, 36 etc. O computador sempre executa um programa, linha a linha, iniciando pela de menor número.

Antes de digitar o próximo programa, digite o seguinte:

NEW <RETURN>

Esse comando apaga qualquer programa que esteja na memória do computador. É aconselhável usar esse comando antes de digitar um novo programa.

Agora digite o seguinte programa. Não se esqueça: a tecla <RETURN> deve ser pressionada após a digitação de cada linha:

```
10 PRINT "QUAL E O SEU NOME?"                <RETURN>
20 INPUT N$                                  <RETURN>
30 PRINT "OLA ";N$;" BEM VINDO AO MSX"       <RETURN>
```

Execute esse programa. Se não ocorreu nenhum erro de digitação você irá ver a linha abaixo na tela:

QUAL É O SEU NOME?
?

A instrução INPUT, na linha 20, está esperando que uma variável string seja digitada. Digite o seu nome e pressione <RETURN>. O computador imprimirá:

OLÁ nome BEM-VINDO AO MSX

Após você digitar o seu nome, o computador executará a linha 30 e dará a mensagem de boas-vindas. Quando o computador executa a última linha ele sabe que alcançou o final do programa e termina a execução dando a você o lembrete "OK". Você agora está de volta ao modo direto.

Se você recebeu uma mensagem de erro em vez de uma mensagem de boas-vindas, cometeu algum erro de digitação em uma ou mais linhas. Antes de procurar os erros, é melhor obter uma listagem do seu programa. Para fazer isso, digite:

LIST e pressione <RETURN> em seguida

O comando irá LISTar o seu programa imediatamente, iniciando pela linha de número menor. Todos os nomes de comandos e variáveis serão LISTados em letras maiúsculas, mesmo que você os tenha digitado em minúsculas. Você deve ver:

```
10 PRINT "QUAL E O SEU NOME?"                 <RETURN>
20 INPUT N$                                   <RETURN>
30 PRINT "OLA ";N$;" BEM VINDO AO MSX"        <RETURN>
```

Agora examine a linha referenciada na mensagem de erro para encontrar os erros de digitação. Uma vez encontrado um erro, modifique essa linha usando as teclas de edição e de cursor descritas no Capítulo 1. Então, com o cursor ainda na linha corrigida, pressione <RETURN>. Isso diz ao computador para substituir a linha antiga pela versão corrigida.

Mova o cursor totalmente para fora da listagem do programa antes de digitar RUN outra vez. Para fazer isso, mantenha a tecla <↓> pressionada até que o cursor esteja na parte inferior da tela. Agora digite RUN outra vez, seguido por <RETURN>. Repita o processo de edição até que o seu programa funcione!

Se a tela parecer muito confusa, pressione <SHIFT> e <CLS> antes de digitar LIST.

Se você tiver cometido muitos erros ao digitar uma linha, às vezes é mais rápido redigitar a linha inteira, ou seja, redigitá-la, em vez de editá-la com as teclas do cursor.

```
10 PRONT "QUAL É O SEU NOME?"
```

Nesse caso, pode ser mais rápido digitar a linha inteira:

```
10 PRINT "QUAL É O SEU NOME?" <RETURN>
```

A linha antiga é então apagada e substituída pela nova.

Se você quiser remover uma linha de seu programa, tudo o que tem a fazer é digitar o número da linha e pressionar <RETURN>. Após ter feito isto, você não pode mais ter a linha de volta, a não ser que a digite de novo:

```
10 <RETURN>
```

apaga a linha 10.

Digite o seguinte programa:

```
10 REM ESTE PROGRAMA SE AUTO-REPETE
20 CLS
30 INPUT "QUAL E O SEU NOME";N$
40 PRINT "OLA ";N$;" BEM VINDO AO MSX"
50 GOTO 30
```

O comando REM diz ao computador para ignorar tudo o que estiver nessa linha. O restante da linha é usado como um lembrete, para você, do que o programa faz. É conveniente incluir comandos REM em seu programa, pois é muito fácil esquecer como um programa funciona quando você voltar a ele posteriormente.

A segunda linha limpa a tela. Você já conhece uma dupla de teclas que faz isso, <SHIFT> <HOME> , mas se quiser limpar a tela durante seu programa deve usar o comando CLS.

O comando na linha 30 já é familiar. Este exemplo mostra como imprimir uma mensagem antes do "?". Tudo o que você tem a fazer é colocar a mensagem entre aspas, seguida por ponto e vírgula e pelo nome da variável. Aqui, a ordem dos símbolos é importante.

A linha 50 diz ao computador para voltar para a linha 30 e continuar a execução a partir dali. Essa manobra é conhecida como "loop infinito", pois o computador volta para a linha 30 sempre que alcança a linha 50. Isso continuará a acontecer até que você interrompa a execução do programa.

Agora tente executar esse programa.

Quando desejar interromper o programa basta pressionar <CTRL> <STOP> , simultaneamente. Você irá obter a mensagem:

```
BREAK in 30
```

Para continuar a execução digite CONT. Não se esqueça de pressionar <RETURN>. Você verá que a execução reinicia de onde foi interrompida.

Se você interrompeu o programa em qualquer outra linha, verá que a execução recomeça na linha seguinte.

EXERCÍCIO

1. Escreva um programa que pergunte seu nome e idade, e imprima-os em seguida.

CAPÍTULO 4

MAIS SOBRE ARITMÉTICA

A esta altura você já conhece os símbolos matemáticos:

\+   −   /   *

Outro símbolo útil é "^". Ele eleva um número a uma dada potência. Por exemplo:

2 elevado a 3 é escrito 2^3

Digite:

PRINT 4^5 <RETURN>

e você obterá a resposta 1024.

Os parênteses podem ser usados nas expressões aritméticas. Qualquer expressão dentro dos parênteses é sempre executada em primeiro lugar.

Todos os símbolos listados abaixo podem ser combinados em uma expressão, sendo a mesma executada na seguinte ordem:

( )	Expressões dentro de ( ) são resolvidas em primeiro lugar.
^	Exponencial.
*, /	Multiplicação e divisão têm a mesma prioridade.
+, −	São calculadas em último lugar.

Por exemplo, o computador executa a expressão abaixo em três estágios:

```
                  (3+4)*7^2
Primeiro estágio  7*7^2
Segundo estágio   7*49
Terceiro estágio  343
```

Você deve incluir o sinal "*" entre os parênteses, ou seja:

```
PRINT (3+4)(4-2) ESTÁ ERRADO!
```

Essa linha deve ser:

```
PRINT (3+4)*(4-2)
```

No exemplo do programa seguinte, é requisitada uma temperatura em graus Fahrenheit, a qual é convertida para graus centígrados:

```
10 REM CONVERSAO DE TEMPERATURAS
20 CLS
30 INPUT "TEMPERATURA FAHRENHEIT";F
40 C = (F-32)*5/9
50 PRINT "TEMPERATURA EQUIVALENTE EM CENTIGRADOS:";C
60 GOTO 30
```

Observe que na linha 40 a instrução LET foi omitida. Essa instrução é opcional. Assim que o computador vê uma variável e, a seguir, um sinal de igual, ele atribui o valor da expressão à direita do "=" à variável numérica do lado esquerdo do "=".

Ao executar esse programa você verá que a temperatura em centígrados é dada em 14 casas decimais. Se você quiser arredondar a temperatura para o valor inteiro pode usar a função INT.

INT converte números reais, isto é, números que contêm casas decimais para números inteiros. O argumento deve ser incluído entre parênteses.

A função INT arredonda um número real para o menor número inteiro mais próximo:

```
 3.4 se torna  3
-2.7 se torna -3
```

Para arredondar um número real para o maior número inteiro mais próximo, e não para o menor inteiro mais próximo, deve-se somar 0.5:

```
INT(3.9)        equivale a 3
INT(3.9+.5)     equivale a 4
```

Você deve indicar ao computador quando quer que ele armazene números inteiros e quando quer que ele armazene números reais. Para distinguir os dois, os nomes de números inteiros terminam com o sinal "%".

Substitua as linhas 40 e 50 do último programa por:

```
40    C%=INT((F-32)*5/9+.5)  <RETURN>
50    PRINT "TEMPERATURA" EQUIVALENTE EM
      CENTÍGRADOS É"; C%   <RETURN>
```

O programa também funcionará se você usar C no lugar de C%, mas o número será armazenado em 14 casas decimais, mesmo que todos os dígitos após o ponto decimal sejam zeros. Os zeros não serão impressos, mas serão armazenados na memória, dessa forma usando um espaço valioso. Assim sendo, é melhor usar nomes de variáveis inteiras sempre que possível.

EXERCÍCIO

1. Converta o programa de temperatura de modo que as temperaturas sejam digitadas em graus centígrados e impressas em Fahrenheit. Você precisará usar a equação:

```
F=C*9/5+32
```

CAPÍTULO 5

O USO DE LOOPS

Este programa imprime qualquer tabela de multiplicação que você desejar. Não se aborreça digitando:

```
10 REM TABUADA DE MULTIPLICACAO
20 INPUT "MULTIPLICADOR";M
30 PRINT "UMA VEZ";M;"=";1*M
40 PRINT "DOIS VEZES";M;"=";2*M
50 PRINT "TRES VEZES";M;"=";3*M
```

e assim por diante. Esse programa é um tanto longo e repetitivo. Uma forma muito mais sensata de fazer o mesmo cálculo é obtida usando-se um loop FOR...NEXT:

```
10 REM TABUADA DE MULTIPLICACAO COM LOOP
20 INPUT "MULTIPLICADOR";M%
30 FOR C%=1 TO 10
40 PRINT C%;"VEZES";M%;"=";C%*M%
50 NEXT C%
60 GOTO 20
```

Observe as variáveis numéricas inteiras que são usadas neste programa: C% e M%.

Na primeira vez que o computador executa a linha 30, ele define C% = 1 e então passa para a linha 40. Quando ele executa a linha 50, o comando NEXT diz para retornar à linha 30. De volta à linha 30, ele incrementa o contador por um, isto é, faz

C% = 2. Este loop é repetido até que C% = 10. Após isso, o computador ignora o comando NEXT da linha 50 e executa a linha 60.

Você pode incrementar o contador em quantidades maiores que 1 usando a palavra STEP no comando FOR...NEXT. Tente o seguinte:

```
10 FOR N=0 TO 12 STEP 2
20 PRINT N
30 NEXT
```

Você deve ver:

```
 0
 2
 4
 6
 8
10
12
```

As restrições para a nomeação do contador são as mesmas que para os nomes das variáveis numéricas. Veja o Capítulo 2.

Se você substituir o limite no programa anterior por 13 ele não terá qualquer efeito sobre os resultados. O loop terminará quando o contador for igual a 12. Com um valor de STEP igual a 2 e iniciando o contador em zero, o mesmo nunca alcançará o valor 13.

Você pode usar tanto números reais como inteiros para definir o valor inicial e final do contador e do tamanho do incremento. Variações negativas também podem ser usadas:

```
10 FOR N= 10 TO 2.5 STEP -2.5
20 PRINT N
30 NEXT N
```

resultando em:

```
10
7.5
5
2.5
```

Seja cuidadoso ao inicializar o contador com um valor maior que o valor final; lembre-se de usar variações negativas ou, de outra forma, ocorrerá uma mensagem de erro.

É possível colocar o loop FOR...NEXT dentro de outro loop FOR...NEXT. Isso é chamado loop em nível. Iremos usar essa técnica no exemplo seguinte para imprimir um desenho com o símbolo "*":

```
10 REM LOOP EM NIVEL
20 PRINT"*"
30 FOR I=1 TO 9
40 FOR J=1 TO I  } UM
50 PRINT "*";    } LOOP
60 NEXT J        } EM NIVEL
70 PRINT "*"
80 NEXT I
```

Você deve ver:

```
*
**
***
****
*****
******
*******
********
*********
```

Observe o " ; " no final do comando PRINT na linha 50. Isso faz com que o comando PRINT seguinte imprima na mesma linha, iniciando na primeira posição vazia, não ocorrendo retorno de cursor.

A linha 30, faz I = 1. Isso limita o contador no loop interno, de modo que na primeira vez que o loop I é executado o "*" é impresso uma só vez. Na vez seguinte da rodada do loop externo, o I é incrementado e passa a valer 2. O J agora pode receber os valores 1 e 2 de modo que o loop interno é executado duas vezes, resultando em dois "*" impressos na mesma linha no loop.

Padrões diferentes podem ser obtidos variando-se o valor do STEP.

EXERCÍCIOS

1. Veja o que acontece se você omitir as linhas 20 e 70.

2. Tente executar o programa anterior com STEP 4 no loop externo. Altere o limite superior desse loop. Com um valor diferente de variação, as linhas 20 e 70 precisarão ser alteradas para formar um padrão regular.

3. Varie o tamanho de STEP no loop interno e veja o que ocorre.

CAPÍTULO 6

O USO DE CONDIÇÕES

Algumas vezes você pode querer que o seu programa tome um determinado caminho dependendo de uma certa condição. O comando **IF...THEN** é usado para isso. Eis um exemplo do seu uso:

```
20  IF A=B THEN GOTO 100
30  PRINT A,B
```

A linha 20 diz ao computador que SE (IF) o conteúdo da variável A for igual ao conteúdo da variável **B, ENTÃO (THEN)** VÁ PARA (GOTO) a linha 100; caso contrário continue com a linha seguinte do programa, que neste caso imprime as variáveis A e B.

Várias condições podem dar continuidade ao comando **IF**:

>	MAIOR QUE
<	MENOR QUE
=	IGUAL A
> =	MAIOR OU IGUAL A
< =	MENOR OU IGUAL A
< >	DIFERENTE DE

Qualquer comando pode vir em seguida do THEN. Por exemplo:

```
20  IF A=B THEN PRINT B
```

O programa seguinte solicita dois números, e então os imprime, o maior em primeiro lugar.

```
10 INPUT"NUMEROS";A,B
20 IF A>B THEN GOTO 40
30 SWAP A,B
40 PRINT"O NUMERO MAIOR E:";A
50 PRINT"O NUMERO MENOR E:";B
```

Neste programa ambos os valores das variáveis são solicitados através do comando INPUT. O computador imprime "Números?" e então espera até que os dois valores tenham sido digitados. Você pode digitar um dos números, uma vírgula, e o segundo número e então pressionar <RETURN>; ou ainda digitar o primeiro número, pressionar <RETURN> e então digitar o segundo número e pressionar <RETURN>. Na segunda opção o computador irá imprimir dois pontos de interrogação após você ter pressionado <RETURN>. O programa continua a ser executado.

Em geral, qualquer quantidade de variáveis pode ser solicitada em um único comando INPUT, desde que as mesmas estejam separadas da mensagem entre aspas por um ponto e vírgula e que uma esteja separada da outra por uma vírgula.

O comando SWAP, na linha 30, troca o conteúdo da variável A pelo conteúdo da variável B. Se a condição A>B, na linha 20, não for verdadeira, então, em vez do programa saltar para a linha 40, ele executa a linha 30. O comando SWAP assegura nesse programa, que o maior número seja colocado na variável A. As duas linhas seguintes imprimem os números. O SWAP também pode ser usado para trocar o conteúdo de variáveis string.

O comando IF...THEN pode incluir a palavra ELSE. Todos os comandos após a palavra ELSE serão executados se a condição antes do THEN não for satisfeita. O programa anterior poderia ser escrito assim:

```
10 INPUT"NUMEROS";A,B
20 IF A>B THEN PRINT "O NUMERO MAIOR E:";A ELSE PRINT
"O NUMERO MAIOR E:";B
30 IF A>B THEN PRINT "O NUMERO MENOR E:";B ELSE PRINT
"O NUMERO MENOR E:";A
```

Pode-se utilizar mais de uma condição após o comando IF usando o AND e o OR. Por exemplo:

```
10  INPUT A,B,C
20  IF A=B AND A > C THEN PRINT A
```

O programa anterior somente irá imprimir o valor de A se ele for igual a B e (AND) maior que C. Compare o programa anterior com o próximo programa:

```
10  INPUT A,B,C
20  IF A=B OR A > C THEN PRINT A
```

Essa versão do programa irá imprimir o valor de A se A for igual a B ou (OR) maior que C.

As desigualdades também podem ser usadas com strings. Uma string é armazenada como uma seqüência de números na memória do computador, sendo cada caractere representado por um número diferente. Para comparar duas strings, o computador inicia o processo comparando dois números representando o primeiro caractere de cada string. Se os dois primeiros caracteres forem iguais, então ele compara os dois seguintes e assim por diante. Caracteres maiúsculos ou minúsculos são representados por números diferentes.

```
De A a Z    pelos números 65 a  90
De a a z    pelos números 97 a 122
```

Outros caracteres como "*" também podem ser comparados. Voltaremos ao assunto futuramente (veja o Capítulo 20).

As strings abaixo estão classificadas em ordem decrescente, ou seja, com a "maior" em primeiro lugar:

```
"zebra"
"aa"
"aA"
"AND"
```

Para resumir o que você aprendeu neste capítulo, eis um curto programa que faz uso completo dos comandos IF, THEN e SWAP.

O programa classifica três números digitados e os exibe em ordem, o maior em primeiro lugar:

```
10 REM CLASSIFICA
20 INPUT "TRES NUMEROS";A,B,C
30 IF B>A THEN SWAP A,B
40 IF C>B THEN SWAP B,C
50 IF B>A THEN SWAP B,A
60 PRINT A,B,C
```

Execute esse programa digitando vários números. Você pode explicar a lógica desse programa?

EXERCÍCIOS

1. Usando o programa anterior, escreva um programa que classifique três strings em ordem alfabética.

2. O programa-exemplo pode trabalhar somente com três números. Você consegue escrever um programa melhor, que possa manusear mais que três números? Um exemplo é fornecido no Capítulo 13.

CAPÍTULO 7

COMANDOS ÚTEIS E DICAS PARA ESCREVER PROGRAMAS

Um comando muito útil para economizar tempo é o comando AUTO. Ao digitá-lo, você descobrirá que a linha número 10 aparece AUTOmaticamente na tela. Digite REM após isso e então pressione <RETURN>. Agora a linha 20 aparecerá. Esse processo continuará indefinidamente. Para parar a numeração AUTOmática de linhas pressione <CTRL> <STOP>.

Antes de digitar um novo programa é aconselhável usar o comando NEW para cancelar as linhas do programa antigo e as suas variáveis. Caso contrário, você poderá descobrir algumas linhas de seu antigo programa misturadas com as do novo, especialmente se o programa anterior for maior que o atual. Isso pode resultar em um caos!

O comando LIST é usado para listar o seu programa na tela. Se você quiser listar apenas uma linha do programa (você pode ter recebido uma mensagem de erro referente a essa linha), digite LIST seguido pelo número da linha:

```
LIST 30
```

Isso faz com que a linha 30 seja exibida na tela.

Ao LISTar o seu programa, você provavelmente notará que os caracteres minúsculos em comandos e funções e nomes de variáveis são substituídos por caracteres maiúsculos.

É sempre aconselhável, dar um título a seu programa usando a instrução REM, de modo a utilizá-la ao longo do programa para explicar cada parte. Isso torna a tarefa bem mais fácil, principalmente quando você voltar a editar o programa após algum tempo da sua criação. O comando REM sempre pode ser removido em uma data posterior caso você ache que está saindo fora dos limites da memória. Assim, tendo em vista que você pode remover um comando REM posteriormente, nunca coloque um comando GOTO fazendo referência a uma linha com REM.

A maneira mais simples de se cancelar uma única linha, é simplesmente digitar o número da linha e <RETURN>. Porém, se você quiser cancelar uma grande parte do programa use o comando DELETE:

```
DELETE 20-80
```

Isso irá cancelar todas as linhas entre os números 20 e 80 inclusive. Seja cuidadoso para não cancelar linhas por acidente, pois, uma vez usado esse comando, a única maneira de se conseguir essas linhas de novo é digitando-as novamente!

Mais de um comando pode ser incluído em uma linha, porém devem ser separados por dois pontos. Por exemplo:

```
20  PRINT "A" :GOTO 10
```

Esta é uma linha válida. A letra "A" será impressa e então o computador irá saltar para a linha 10. Qualquer quantidade de comandos pode ser ligada dessa maneira. Isso irá economizar memória do computador, pois uma certa quantidade de memória é usada para armazenar o número de cada linha. Entretanto, um programa é geralmente mais legível se você evitar o uso de muitos comandos na mesma linha. Existe também um limite no número de caracteres que podem ser incluídos em uma mesma linha – até 255 caracteres podem ser digitados, incluindo os espaços. O computador ignora qualquer caractere além desse limite.

Para demonstrar isso, digite o seguinte:

```
10  PRINT "**********..."
```

preenchendo a tela com "*"s. Agora pressione <SHIFT> <CLS> e execute o programa; somente 247 "*" serão impressos. O computador armazenou a linha até o caractere número 255. Observe que se você esquecer as aspas finais de uma string, o computador

automaticamente assume a sua presença. Entretanto, não adquira o hábito de fazer isso, ou poderá produzir confusão.

Mais um detalhe acerca do PRINT – existe uma abreviação para indicá-lo: é o "?". Digite:

```
10 ?
```

Agora digite LIST. O ponto de interrogação será substituído por PRINT.

Ao inspecionar um programa listado, você pode descobrir que se esqueceu de digitar uma linha, ou mesmo um determinado número de linhas. E possível que não haja espaço suficiente para inserir as linhas esquecidas devido ao espacejamento inadequado de números de linhas. Se isso acontecer, você pode RENUMerar as linhas do seu programa usando o comando RENUM. Por exemplo:

```
10 REM
11 REM
12 REM
```

Para inserir uma linha entre as linhas 10 e 11, digite RENUM. Agora LISTe o seu programa e você verá:

```
10 REM
20 REM
30 REM
```

Agora você pode inserir facilmente novas linhas entre as linhas 10 e 20, usando números de linha entre 11 e 19.

É possível armazenar mais de um programa na memória. Para fazer isso você pode armazenar o primeiro programa entre as linhas de número 10 a 100, por exemplo, e o segundo entre as linhas 500 e 1000. Quando digitar RUN você deve evitar que o computador execute o primeiro programa e vá direto ao segundo. Para fazer isso, use o comando END. Essa deve ser a última instrução do primeiro programa. Ao encontrar o END o micro pára a execução e coloca você de volta no modo direto.

Para continuar a execução que, neste caso, significa executar o segundo programa armazenado na memória, você deve digitar CONT seguido por <RETURN>.

Uma forma de executar o segundo programa antes do primeiro, é ir diretamente a ele, usando:

```
RUN 500
```

Isso inicia a execução na linha 500. O mesmo resultado pode ser obtido usando:

```
GOTO 500
```

O comando STOP também pode ser usado em um programa. Esse comando, da mesma forma que um comando END, coloca o micro de volta ao modo direto. Entretanto, ele produz a mensagem "break in...", informando o número da linha do comando STOP. Após um comando STOP você pode continuar a execução do programa digitando CONT <RETURN>.

Você pode parar a execução de um programa a qualquer momento pressionando a tecla <STOP>. Para continuar a execução pressione essa tecla outra vez. Você não será capaz de usar o comando CONT neste caso, pois ele só pode ser usado com comandos STOP dentro do programa. Contudo, se você pressionar <CTRL> <STOP>, interromperá o programa e voltará ao modo direto. Uma vez no modo direto, você pode reiniciar a execução a partir da próxima linha, usando o comando CONT. Digite o seguinte programa:

```
10 REM
20 PRINT"adeus"
30 GOTO 20
```

e execute-o em seguida. Agora pressione <STOP>. Isso irá parar o programa. A única forma de reiniciar a execução é pressionar a tecla <STOP> novamente. Execute o programa de novo, desta vez parando-o pressionando <CTRL> <STOP>. Reinicie o programa usando CONT.

Agora digite este programa:

```
10 REM
20 PRINT "OLA"
30 STOP
40 PRINT "ADEUS"
50 END
60 GOTO 20
```

Execute-o!! Use o comando CONT tanto após o comando END como após o comando STOP para reiniciar a execução do programa.

Um comando diferente, que você não precisará usar neste momento, mas que é bom conhecer, é o CLEAR. Ele diz ao computador para apagar todas as variáveis até então armazenadas na memória. Ele também pode ser usado para conseguir espaço para uma variável. Ao se ligar o computador, as strings podem ocupar somente 200 caracteres na memória do micro. Por exemplo, você pode aumentar isso para 255 caracteres digitando:

```
CLEAR 255
```

Outro comando relacionado com a memória do computador é o FRE. Ele indica quanta memória livre existe tanto na área do programa BASIC como na área de armazenamento de strings:

PRINT FRE(0)	Fornece a memória livre na área do programa BASIC.
PRINT FRE("")	Fornece a memória livre na área de string.

Finalmente, dois comandos que foram projetados para ajudar você a descobrir erros em seus programas. TRON, que significa "Trace On", faz o computador exibir o número de cada linha à medida que ela é executada. Digite:

```
10 REM USO DO TRON
20 PRINT "OLA, ESTA LEVANTADO?"
30 INPUT "SIM OU NAO";A$
40 IF A$="NAO" THEN BEEP: GOTO 20
50 PRINT "VA PARA A CAMA"
```

Agora digite TRON e execute o programa. Aparecerá o seguinte:

```
[10][20]    OLÁ, ESTÁ LEVANTADO?
[30]        SIM ou NÃO?
```

Aqui o programa espera por uma entrada de dados. Se você digitar SIM, irá receber:

```
[40][50]    VÁ PARA A CAMA
```

Se digitar NÃO, você irá receber:

```
[40][20]    OLÁ, ESTÁ LEVANTADO?
[30]        SIM ou NÃO?
```

O comando BEEP, na linha 40, é o comando mais fácil de se usar para se obter um som no computador.

Para desativar o TRON, use TROFF.

CAPÍTULO 8

TECLAS DE FUNÇÃO

São as cinco teclas situadas na parte superior do teclado. Cada tecla tem duas funções, uma das quais é utilizada pressionando-se a tecla diretamente, e a outra pressionando <SHIFT> e a tecla simultaneamente; ou seja, as funções F1, F2, F3, F4 e F5 são usadas pressionando-se a própria tecla, enquanto que F6, F7, F8, F9 e F10 se obtêm pressionando-se <SHIFT> mais a tecla correspondente.

Ao se ligar o computador, as teclas já estão programadas para certas funções. Uma lista dessas funções pode ser exibida digitando-se o comando:

```
KEY LIST
```

O conteúdo das cinco primeiras teclas é visualizado na última linha da tela. As outras cinco são exibidas, ao se pressionar <SHIFT>. Para desligar essa visualização digite KEY OFF, e para ligar a visualização digite KEY ON. Os últimos três comandos podem ser incluídos em um programa.

No momento, esqueça as teclas F1, F6 e F7. Falaremos sobre elas posteriormente. As funções das outras sete teclas devem ser familiares, pois elas são muito úteis:

Tecla	Exibição	Descrição
F1	color	Veja o Capítulo 22.
F2	auto	Esta tecla é programada para imprimir AUTO na tela, seguido de um espaço, ou seja, esta tecla é o equivalente a digitar "AUTO".
F3	goto	É o equivalente a se digitar GOTO seguido de um espaço.
F4	list	É o equivalente a se digitar LIST seguido de um espaço.
F5	run	A função desta tecla é ligeiramente diferente das anteriores. Ela é o equivalente a digitar RUN seguido de <RETURN>. Esta tecla executa o programa que estiver contido na memória do micro.
F6	color 15,11*	Veja o Capítulo 22.
F7	cload"	Veja o Capítulo 11.
F8	cont	É o equivalente a CONT seguido por <RETURN>. Se você parou um programa pressionando <CTRL> <STOP> ou com uma instrução END ou STOP, pode reiniciar a execução a partir da linha seguinte pressionando <SHIFT> e F8.
F9	list+CHR$(13)**	F9 lista o programa automaticamente.
F10	run	Esta tecla de função é útil pois limpa a tela e executa qualquer programa que estiver na memória.

* No HOT-BIT temos COLOR 15,4,4.

** No HOT-BIT temos LIST, que lista a última linha editada.

É possível redefinir as funções destas teclas usando o comando KEY, seguido pelo número da tecla, uma vírgula e, então, uma string contendo a nova função. Digite:

```
KEY 1,"PRINT" <RETURN>
```

Isso fará com que o comando PRINT seja exibido sempre que F1 for pressionada.

Observe que a lista na base do modo de tela mudou: COLOR foi substituído por PRINT.

```
KEY 6, "NEW" + CHR$(13)
```

programa a tecla 6 para executar o comando NEW. CHR$(13) é o equivalente a se pressionar <RETURN> (veja maiores detalhes no Capítulo 20).

Quando definir uma tecla, você pode usar um máximo de 15 caracteres na string. Observe que CHR$(13) ocupa a posição de um caractere.

EXERCÍCIO

1. Programe a tecla F7 para RENUMerar o seu programa imediatamente após ter sido pressionada.

CAPÍTULO 9

MAIS SOBRE O COMANDO PRINT E O MODO DE TELA

Até agora você usou o comando PRINT para imprimir strings e variáveis com uma vírgula ou um ponto e vírgula separando a lista de itens a serem impressos. O ponto e vírgula faz com que os itens sejam impressos próximos uns aos outros; a vírgula faz com que dois itens sejam impressos na mesma linha, o primeiro no início da linha e o segundo separado por 15 espaços.

Um ponto e vírgula no final de um comando PRINT suprime o retorno do cursor de modo que o próximo comando PRINT irá iniciar a imprimir na mesma linha:

```
10 PRINT "SER OU NAO SER-";
20 PRINT "EIS A QUESTAO"
```

Tudo isso é impresso na mesma linha:

```
SER OU NÃO SER – EIS A QUESTÃO
```

Você provavelmente descobrirá que em determinado estágio você irá querer imprimir uma tabela de resultados, ou talvez uma lista das opções já tabuladas. Dois comandos que o ajudam a fazer isso são TAB e LOCATE.

Para saber usar esses comandos você deve conhecer os diferentes modos de tela.

Existem dois modos de tela de texto, SCREEN 0 e SCREEN 1. Ao se ligar o computador, você está automaticamente em um desses modos. Neste capítulo será usado o modo padrão: SCREEN 1. Para ter certeza de que você está nesse modo, digite:

```
SCREEN 1 <RETURN>
```

A tela do computador, nesse modo, tem 32 posições de largura por 24 de altura. Cada posição de caractere ocupa um quadrado, de oito por oito pontos.

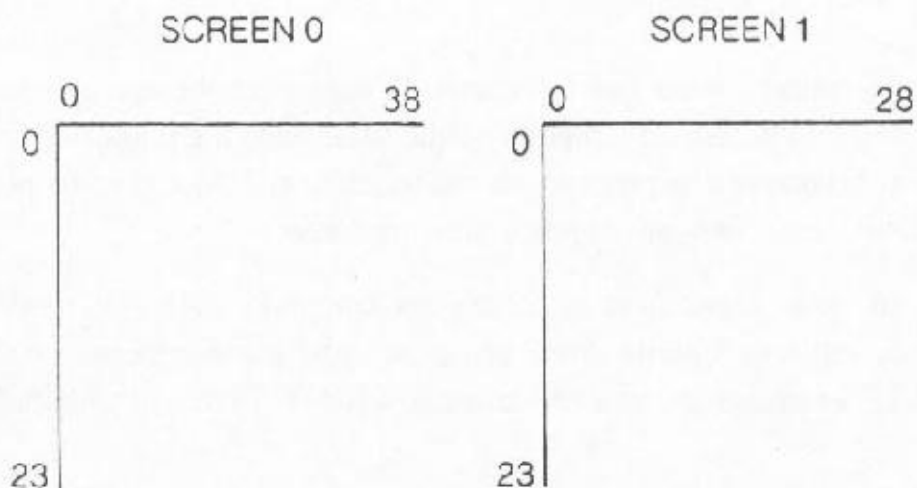
O outro modo de tela de texto, SCREEN 0, tem 40 posições de largura por 24 de altura. As posições de caractere nesse modo são ligeiramente menores, cada uma ocupando um retângulo de seis por oito pontos. Nesse modo de tela a margem tem a mesma cor do fundo. Para acessar esse modo de tela digite:

```
SCREEN 0 <RETURN>
```

Agora retorne ao SCREEN 1 digitando:

```
SCREEN 1 <RETURN>
```

A origem das coordenadas de ambos os modos de tela de texto é o canto superior esquerdo do modo de tela. Observe que a primeira linha e a primeira coluna de cada tela são chamadas linha 0 e coluna 0, respectivamente. Dessa forma, a última coluna do SCREEN 0 é a coluna 39 e no SCREEN 1 é a coluna 31. A última linha em ambos os modos de tela é a linha 23.

Você não pode imprimir na última linha antes de usar KEY OFF para apagar a lista de teclas de função.

Você pode alterar o número máximo padrão de caracteres em cada linha usando o comando WIDTH, seguido de um número. Esse número define a largura máxima da tela. Na tela do SCREEN 0, o número máximo de caracteres que você pode definir ao longo do modo de tela é 40 e na tela do SCREEN 1 é 32. Você pode reduzir a largura de qualquer um dos dois modos de tela para somente 1 caractere se desejar! Experimente:

```
SCREEN 0    <RETURN>
WIDTH  20   <RETURN>
```

Esses comandos definem a largura máxima da tela no SCREEN 0 para 20 caracteres. Você agora vai descobrir que só pode escrever nas 20 colunas mais centrais. Para voltar à largura e tela padrões, digite:

```
SCREEN 0    <RETURN>
WIDTH  40   <RETURN>
```

Agora voltemos à tabulação! O comando TAB é sempre usado em conjunto com PRINT. Esse comando faz com que o computador comece a imprimir na coluna especificada (ou coordenada x) pelo comando TAB. Por exemplo:

```
PRINT TAB(10)      "Isto faz iniciar a impressão na 11ª posição de caractere."
```

Não se esqueça de que a primeira posição de caractere da tela é a coluna 0, linha 0. Alternativamente, você pode imaginar como sendo a posição de coordenada de caractere x, y = (0,0).

Você pode usar mais de um comando TAB em um comando PRINT:

```
PRINT TAB(3)4      TAB(7)8
```

Isso irá imprimir o número 4 na quarta coluna. O espaço em branco que sempre precede um número positivo é impresso na terceira coluna. O número 8 é impresso na mesma linha, com o espaço em branco que o precede na sétima coluna. Não é preciso pontuação entre os comandos TAB, mas não se esqueça dos parênteses.

Você só pode especificar a coluna no comando TAB, de modo que se quiser imprimir tanto em uma coluna como em uma linha em particular, você deve usar o comando LOCATE exatamente antes do comando PRINT. Esse comando define a posição

do cursor; o primeiro número no comando LOCATE é o número da coluna (ou coordenada x); o segundo número é o da linha (ou coordenada y). Por exemplo:

```
1 CLS
10 LOCATE 6,2
20 PRINT "NOME"
30 LOCATE 20,2
40 PRINT "APELIDO"
```

que imprime:

```
     NOME          APELIDO
```

na terceira linha da tela.

Você precisa de um comando LOCATE para cada comando PRINT. Uma alternativa é usar o LOCATE somente para definir a linha, e então usar PRINT TAB. O programa seguinte fornece o mesmo resultado que o anterior:

```
10 CLS
20 LOCATE ,2
30 PRINT TAB(6)"NOME"TAB(20)"APELIDO"
```

Quando a coordenada x é deixada de fora do comando LOCATE, o computador entende que você quer permanecer na posição x atual. O comando CLS da linha 10 está definido para x = 0, de modo que esse valor de x é usado para posicionar o cursor na linha 20. Observe que, apesar de você não ter especificado a coordenada x, você tem de escrever a vírgula.

O LOCATE não pode ser usado no modo direto porque, assim que esse comando é executado nesse modo, o cursor é colocado no lado esquerdo da linha seguinte, pronto para se digitar um novo comando.

Você pode usar o LOCATE para desligar o cursor de dentro de um programa, isto é, desativar o pequeno quadrado branco. Para fazer isso, coloque uma vírgula após a coordenada y e siga-a com um zero. Para ativar o cursor novamente, substitua o 0 por 1. Por exemplo:

```
10 LOCATE ,,0
```

desativa o cursor. Enquanto que:

```
10 LOCATE 10,10,1
```

ativa o cursor e move-o para a posição de coordenada (10,10). Esse efeito não pode ser obtido no modo direto.

Agora vamos a outro comando, SPC. Esse comando é usado dentro de um comando PRINT para especificar o número de espaços a serem impressos. O pequeno programa que você digitou anteriormente, para imprimir os cabeçalhos "NOME" e "APELIDO", pode ser substituído por:

```
10 CLS
20 LOCATE ,2
30 PRINT TAB(6)"NOME"SPC(8)"APELIDO"
```

Qualquer número inteiro entre 1 e 255 pode seguir ao comando SPC; o número deve ser colocado entre parênteses. Sempre que você quiser imprimir uma string que contenha muitos espaços, é aconselhável usar o comando SPC, pois ele economiza memória.

Existe um método alternativo de se imprimir espaços, usando SPACE$. Esse comando é mais versátil que o SPC, pois pode ser usado fora do comando PRINT, enquanto que SPC não. SPACE$ é usado para formar strings inteiras de espaços. Por exemplo:

```
A$=SPACE$(10)
```

é o equivalente a:

```
A$="          "
```

Usando esse comando, podemos atribuir nossos cabeçalhos a uma variável e imprimi-los tantas vezes quantas desejadas:

```
10 H$ = "NOME"+SPACE$(8)+"APELIDO"
20 CLS
30 LOCATE ,2
40 PRINT TAB(6)H$
```

Não se esqueça de que isso não pode ser feito com o SPC.

```
A=SPC(10) É ERRADO!
```

EXERCÍCIO

1. Escreva um programa que entre a data de hoje e a data de seu nascimento; agora imprima a sua idade. Tabule as mensagens impressas na tela usando LOCATE.

CAPÍTULO 10

PROGRAMAÇÃO INTERATIVA

Um programa é chamado "interativo" se, em algum ponto durante a sua execução, ele receber dados do usuário. Este capítulo tem a finalidade de tratar dos comandos INPUT, INKEY$ e INPUT$, usados em programas interativos.

Apesar de TAB poder ser usado somente com o comando PRINT, LOCATE pode ser usado com o comando INPUT também:

```
10 LOCATE 10,22
20 INPUT "ARTIGO, VALOR";A$,V
```

Você irá ver:

```
ARTIGO, VALOR?
```

aparecer na linha 22 da tela. O computador irá esperar até que você tenha INSERIDO uma string e então um número. Para fazer isso, você pode pressionar <RETURN> após ter digitado a variável string e então inserir o número, ou digitá-los juntos, separando a string e o número por uma vírgula. Entretanto, se usar o comando LINE INPUT, você pode digitar a linha inteira de texto em uma única variável string!

```
10 LOCATE 10,22
20 LINE INPUT "ARTIGO, VALOR";L$
```

Você irá ver:

ARTIGO, VALOR?

na linha 22 da tela. Agora, se você digitou:

Bananas, amendoins e saquinhos de chá Cz$25,00

a linha inteira, incluindo os brancos, será colocada na variável L$. Se você substituir LINE INPUT por um comando INPUT, tudo após a vírgula será ignorado.

LINE INPUT pode ser usado somente com variáveis string de modo que:

10 LINE INPUT A É ERRADO!

A linha que você digitou pode ter qualquer tamanho até 200 caracteres. O tamanho da linha pode ser estendido para até 255 caracteres, usando o comando CLEAR (veja o Capítulo 7).

Ao contrário de INPUT, LINE INPUT não produz automaticamente o lembrete do ponto de interrogação, de modo que se você quiser isso, deve incluí-lo na mensagem após o comando.

Em determinado ponto do programa, você pode querer que o computador pare e pergunte sobre as diversas opções que você queira escolher. Por exemplo, em um programa de jogo, você pode querer que o computador pergunte ao jogador se ele quer que as instruções sejam exibidas ou, no final do jogo, se ele quer jogar outra vez. Você pode usar o comando INKEY$ para fazer isso. Esse comando testa o teclado para ver se uma tecla foi pressionada. Se uma delas o foi, o caractere da tecla é introduzido na variável string INKEY$; se não, uma string vazia é retornada para a variável string INKEY$ e o computador executa a linha seguinte do programa.

A seguinte listagem pode ser colocada no final de um jogo:

```
100 CLS
110 LOCATE ,10
120 PRINT "CONTINUA JOGANDO ?"
130 PRINT "TECLE S (SIM) OU"
140 PRINT "         N (NAO)"
150 A$=INKEY$
160 IF A$="N" OR A$="n" THEN END
170 IF A$="S" OR A$="s" THEN GOTO 100
180 GOTO 150
```

Na primeira vez que a linha 150 é alcançada, INKEY$ ou contém uma string vazia ou o caractere de uma tecla sendo pressionada. Se uma das teclas, S, s, N ou n foi pressionada, enquanto o computador estava na linha 150, então o computador termina o programa ou inicia-o a partir da linha 10. O comando GOTO na linha 180 faz o computador voltar para a linha 150 até que uma das quatro teclas de caractere ("S", "s", "N" ou "n") reconhecidas pelo computador seja pressionada.

Não se esqueça de que INKEY$ é uma string e apenas pode ser comparada a outra variável string de modo que:

A=INKEY$ É ERRADO!

e resultará em um erro do tipo "type mismatch" no Expert ou "tipo desigual" no HOT-BIT:

Outra função similar à INKEY$ é INPUT$. Da mesma forma que INKEY$, ela lê o teclado direamente, mas em vez de somente ler, um caractere ela lê o número de caracteres especificados no comando INPUT$, isto é:

```
10 A$=INPUT$(5)
20 PRINT A$
```

O computador irá esperar na linha 10 até que cinco teclas tenham sido pressionadas. A linha 20 então imprime esses caracteres. O número seguinte ao INPUT$ deve ser um inteiro na faixa de 1 a 200. Se um número real desta faixa for usado, ele será trocado para o inteiro menor mais próximo.

EXERCÍCIO

1. Use um comando INPUT no início de um programa para digitar uma senha. As primeiras linhas de um programa podem perguntar pela senha e a execução do restante do programa deve iniciar somente após a senha correta ter sido digitada.

CAPÍTULO 11

GRAVANDO O SEU PROGRAMA EM UMA FITA

Neste estágio, você já pode escrever programas bastante longos. Em vez de digitá-los toda vez que quiser executá-los, você vai descobrir que é muito mais conveniente armazenar o seu programa em fita.

Em primeiro lugar, conecte o seu cassete ao computador. A maioria dos cassetes serve; um simples gravador monofônico é suficiente. É muito útil ter um contador de fita para anotar onde você armazenou o programa. O cassete deve ter uma tomada de entrada para usar com o microfone e uma tomada de saída para usar com fones de ouvido.

Ligue o cassete ao seu micro seguindo as instruções do manual do fabricante.

Coloque uma fita virgem no seu cassete. Rebobine-a se necessário e então execute-a até que alcance o final da parte transparente no início da fita. Zere o contador de voltas.

Agora digite um pequeno programa em seu computador e, então, acrescente:

```
CSAVE"nome" <RETURN>
```

Você pode dar ao seu programa qualquer nome que desejar, desde que não exceda a seis caracteres. Os caracteres podem ser letras maiúsculas ou minúsculas, ou números. O primeiro caractere do nome deve ser uma letra.

Agora pressione os botões RECORD e PLAY do seu cassete e então pressione <RETURN> no computador.

Quando o computador terminar de gravar o programa, ele emitirá uma mensagem "OK". Pare a fita e anote o número do contador de fita.

Para verificar se você gravou o seu programa com sucesso, desligue o computador para apagar o programa da memória. Rebobine a fita para o início ou na posição zero do contador. Agora ligue o computador e digite:

```
CLOAD"nome"  <RETURN>
```

ou somente:

```
CLOAD  <RETURN>
```

Se o nome do programa não for especificado, o computador carregará o primeiro programa que encontrar na fita. Agora pressione o botão PLAY no cassete. Você deve receber a seguinte mensagem na tela assim que o computador tiver encontrado o programa no Expert:

```
FOUND: nome   (no HOT-BIT é ACHEI: nome)
```

quando o programa tiver sido carregado você receberá a mensagem "OK". Assim que ela aparecer, pare a fita.

Observe que a tecla de função F7 é programada para imprimir "CLOAD" na tela. Usando-a, você somente precisa digitar o nome do programa e as aspas finais.

Você pode verificar que o programa no computador é o mesmo que o programa da fita, isto é, não ocorreram erros de gravação ou carga, pelo uso do comando CLOAD? Para usá-lo, rebobine a fita até o início do programa e digite:

```
CLOAD?"NOME"  <RETURN>
```

Ligue a fita.

Se o programa da memória for o mesmo que o programa da fita você irá receber uma mensagem "OK". Se não a receber, você não conseguiu carregar o programa com sucesso (veja a seção de problemas no final deste capítulo).

Quando armazenar diversos programas em uma fita é conveniente deixar espaços entre eles, bem como anotar a leitura do contador de fita no início de cada programa. Se você gravar um programa em cima de outro, perderá o primeiro programa.

Suponha que você tenha dois programas gravados na fita: o primeiro chamado "plot" e o segundo "ponto". Se você digitar:

```
CLOAD"ponto"
```

e ler a fita a partir do início, receberá as seguintes mensagens no caso do EXPERT: (*)

```
CLOAD"ponto"
SKIP :plot
Found:ponto
OK
```

(*) No caso de HOT-BIT as mensagens serão:

```
PULEI : plot
ACHEI : ponto
```

O computador diz que pulou o programa "plot" e então carregou "ponto".

Agora vamos ver a função da tomada preta. Você pode usá-la se tiver uma tomada de controle remoto em seu cassete, marcado com "REMOTE". Neste caso, conecte-o na tomada preta e digite:

```
MOTOR ON   <RETURN>
```

Se agora você introduzir CLOAD/CLOAD? ou CSAVE, o motor do cassete será automaticamente desligado pelo computador quando este terminar de ler ou de gravar um programa da fita. Isso é bastante útil, pois é muito comum esquecer de desligar o cassete após o uso de CLOAD ou CSAVE.

Para desligar o motor do cassete, digite:

```
MOTOR OFF   <RETURN>
```

ou digitando somente:

```
MOTOR   <RETURN>
```

que terá o mesmo efeito de uma chave de controle que desliga se estiver ligado e liga se estiver desligado.

PROBLEMAS COM O CASSETE

Se você está tendo problemas ao gravar ou carregar os seus programas, verifique:

1. Se você conectou o cassete ao computador.

2. Se você conectou corretamente as tomadas ou se não estão desconectadas.

3. Tente variar os controles de volume e tom do cassete. Normalmente, é melhor deixar o controle do tom no máximo e somente variar o controle de volume. Inicie no volume médio e tente aumentar ou diminuir o volume a partir daí.

4. Verifique se você está escrevendo o nome do programa de forma correta, caso você não consiga carregar um programa que já gravou.

Se após todas essas tentativas você ainda não conseguir carregar o programa, é provável que haja um defeito físico na sua fita cassete.

CAPÍTULO 12

LENDO DADOS EM MATRIZES

Até agora você somente digitou um número limitado de itens por vez. Mas existe um limite quando você quer gravar um grande volume de dados, talvez uma longa lista de nomes ou números. Suponha que você queira manter uma agenda telefônica de todas as pessoas que conhece. Tente o seguinte:

```
10 REM LISTA TELEFONICA VERSAO 1
20 INPUT "NOME E TELEFONE";N1$,NUM1
30 INPUT "NOME E TELEFONE";N2$,NUM2
40 INPUT "NOME E TELEFONE";N3$,NUM3
```

e assim por diante, mas com apenas dez nomes esse programa já se tornará grande e repetitivo. É muito melhor usar um loop e executá-lo a cada entrada na agenda. Para isso, você precisará usar uma matriz.

Uma matriz é um grupo de variáveis do mesmo tipo, mas diferem umas das outras pelo seu índice.

A(n) é uma matriz numérica. Se n = 10, então essa matriz pode aceitar até 11 variáveis chamadas A(0), A(1), A(2), ..., A(10). Você pode pensar que A é um nome de rua, caso em que o número a seguir forneça o número da casa na rua.

Se você quiser usar matrizes com mais de onze elementos, deve informar ao computador o tamanho da matriz e seu nome. Isso é feito usando o comando DIM:

```
DIM NUM(12)
```

Trata-se de uma matriz numérica unidimensional, chamada NUM, que tem treze elementos, NUM(0), NUM(1), ..., NUM(12). A matriz é chamada unidimensional, pois ela pode ser representada por um diagrama unidimensional.

```
NUM(0)
NUM(1)
NUM(2)
NUM(3)
NUM(4)
NUM(5)
NUM(6)
NUM(7)
NUM(8)
NUM(9)
NUM(10)
NUM(11)
NUM(12)
```

O primeiro caractere de um nome de matriz deve ser uma letra do alfabeto e o restante podem ser letras ou números. Da mesma forma que com variáveis simples, comandos ou caracteres de pontuação não podem ser incluídos no nome. Somente os dois primeiros caracteres são reconhecidos pelo computador e caracteres minúsculos não são distinguidos de maiúsculos. Assim, os nomes a seguir têm o mesmo valor para o computador:

```
ab(10)
AB(10)
Abaco(10)
```

Se você quiser ler strings em uma matriz, deve usar uma Matriz de String. Aparentemente a única diferença entre ela e a Matriz Numérica é que o nome deve terminar com o sinal "$" (da mesma forma que as variáveis string).

Nomes permitidos incluem:

```
JOGO$(20)
ENDEREÇO$(10)
A1$(5)
```

Estes nomes não serão permitidos:

TOMO$(12)	Inclui o comanto TO.
QUESTÕES?$(6)	Sinais de pontuação não são permitidos.
FINAME$(200)	NAME é uma palavra reservada.

Você pode dimensionar em um comando no início do programa, todas as matrizes que quiser usar:

```
DIM A$(12),B(13),C(20)
```

Não dimensione mais espaço de matriz que o necessário, pois ao receber um comando DIM o computador define as seções de memória à parte para cada matriz, de modo que se você não usar todas as matrizes, estará desperdiçando memória.

Você pode economizar memória usando matrizes inteiras da mesma forma que matrizes de números reais, isto é, NUM%(12) irá aceitar até 13 variáveis inteiras, nos elementos NUM%(0), NUM%(1), ..., NUM%(12).

As matrizes multidimensionais serão descritas no próximo capítulo.

Você agora pode ler os nomes e números da agenda telefônica, usando um loop:

```
10 REM LISTA TELEFONICA VERSAO 2
20 REM DADOS NAS MATRIZES N$ e NUM
30 CLS
40 INPUT "NUMERO DE ENTRADAS";E
50 N=E-1
60 DIM N$(N),NUM(N)
70 FOR I=0 TO N
80 CLS
90 LOCATE ,10
100 INPUT "NOME     ";N$(I)
110 INPUT "TELEFONE";NUM(I)
120 NEXT I
```

A variável N é usada no comando DIM da linha 40 para assegurar que haja o mesmo número de elementos nas matrizes N$ e NUM à medida que as entradas na agenda são feitas. N é um a menos que E, já que o primeiro elemento de cada matriz é o elemento (0) e não o elemento (1).

Todos os nomes e números de telefones foram digitados em duas matrizes. A parte seguinte do programa exibe a agenda. O programa agora parece assim:

```
10 REM LISTA TELEFONICA VERSAO 3
20 REM DADOS NAS MATRIZES N$ e NUM
30 CLS
40 INPUT "NUMERO DE ENTRADAS";E
50 N=E-1
60 DIM N$(N),NUM(N)
70 FOR I=0 TO N
80 CLS
90 LOCATE ,10
100 INPUT "NOME        ";N$(I)
110 INPUT "TELEFONE";NUM(I)
120 NEXT I
130 REM  MOSTRA A LISTA
140 CLS
150 PRINT TAB(2)"NOME"TAB(18)"TELEFONE"
160 PRINT TAB(2)"*************************"
170 FOR I=0 TO N
180 LOCATE 2,4+I*2
190 PRINT N$(I)
200 LOCATE 18,4+I*2
210 PRINT NUM(I)
220 NEXT I
```

Quando o programa termina, você perde todos os nomes e números! Porém, é possível gravar os dados no final do programa. Você pode então armazenar os dados em fita com o programa.

Para fazer isso, use os comandos READ e DATA.

Um comando DATA pode ser seguido por uma longa lista de números ou dados seqüenciais, com os itens de dados separados por vírgula:

```
DATA    Tomas,4402073,Henrique,2246607,Gerson,4421708
```

A única restrição ao número de itens de dados armazenados é que o número total de caracteres não pode exceder a 255.

Você pode ter tantos comandos DATA quantos quiser em um programa. Eles podem ser armazenados no lugar que você quiser, pois o computador irá saltar sobre os

comandos DATA até que apareça um comando READ. Freqüentemente, é melhor manter todos os comandos DATA no final do programa, pois dessa forma é mais fácil alterar os comandos DATA sem ter de procurá-los por todo o programa.

Vamos voltar ao programa da agenda. Agora, em vez de entrar com os dados enquanto o programa estiver rodando, a última versão do programa já contém os dados nos comandos DATA. Nós sabemos o número de entradas da agenda, já que as colocamos nos comandos DATA, de modo que não precisamos inserir o número de entradas. O número de entradas é colocado na variável E, já que você pode querer alterar o número de entradas em uma data posterior e dessa forma precisamos alterar esse número. Neste caso, somente a linha 30 terá de ser alterada. O programa agora fica assim:

```
10 REM LISTA TELEFONICA VERSAO 4
20 REM LE OS DATA E GUARDA NAS MATRIZES
30 E=4
40 N=E-1
50 DIM N$(N),NUM(N)
60 CLS
70 REM MOSTRA OS DATA
80 PRINT TAB(2)"NOME"TAB(18)"TELEFONE"
90 FOR I=0 TO N
95 READ N$(I),NUM(I)
110 LOCATE 2,4+I*2
120 PRINT N$(I)
130 LOCATE 18,4+I*2
140 PRINT NUM(I)
150 NEXT I
160 END
170 DATA JOAO,8142233, CARLOS ,3332222
180 DATA "CARLOTA",7773321, PEDRO ,12 555
```

Quando o primeiro comando READ for encontrado, o computador lerá o primeiro item do primeiro comando DATA para a variável do comando READ. Neste caso, TOMAS é lido para o elemento N$(0) da matriz N$.

Na segunda vez que um comando READ for encontrado, o segundo item do comando DATA é READ. Uma vez que todos os itens em um comando DATA tenham sido lidos, o computador vai para o comando DATA seguinte e continua a ler seqüencialmente.

Você pode achar desnecessário ler o dado em uma matriz neste programa. Você está certo! Entretanto, uma vez que os dados estão em uma matriz, nós podemos processá-

la, se desejarmos, antes de exibi-la. O capítulo seguinte fornece a versão final deste programa, na qual as entradas são classificadas em ordem alfabética antes de serem exibidas.

Você pode usar um comando READ para ler mais de um dado colocado em um comando DATA. O comando READ seguinte inicia no primeiro item de dados não lido. Por exemplo:

```
10 READ A,B
20 READ A$,B$
30 PRINT A;B;A$;B$
40 DATA 10,20,"Isto e' uma prova.", OLA
```

resulta em:

```
10  20  Isto é uma prova. OLÁ.
```

Seja cuidadoso quando usar comandos DATA. Você deve se assegurar de que a ordem dos dados nos comandos correspondem à ordem dos tipos de variáveis do comando READ. Se você estiver lendo para uma variável numérica, você deve ter um número na posição correspondente do comando DATA.

Você não pode colocar variáveis em comandos DATA, somente strings e números. Strings, em comandos DATA, não precisam ser incluídas entre aspas, a menos que contenham vírgulas, dois pontos ou espaços.

Pode ser que você queira usar o mesmo comando DATA duas vezes. Na primeira vez, você pode querer apenas imprimir os dados, e na segunda fazer alguns cálculos com eles. Para fazer com que o computador volte para um comando DATA que ele já tenha lido, você deve usar o comando RESTORE.

RESTORE faz com que o computador volte a ler os dados desde o primeiro comando DATA. RESTORE, seguido por um número de linha, faz com que o computador volte a ler desde o primeiro comando DATA nessa ou na linha que segue esse número. Assegure-se de que você colocou um número de linha válido após o RESTORE,

isto é, um que exista no programa. Execute o seguinte programa:

```
10 READ A,B
20 PRINT "A=";A,"B=";B
30 RESTORE
40 READ C,D
50 PRINT "C=";C,"D=";D
60 DATA 10,20
```

Você verá:

```
A=10    B=20
C=10    D=20
```

CAPÍTULO 13

MANUSEIO DE DADOS E SUA ORDENAÇÃO

No capítulo anterior, fizemos um programa de agenda telefônica; seria ótimo imprimir a agenda em ordem alfabética. Nós já vimos um método limitado de classificar listas em ordem numérica no Capítulo 6. Entretanto, o método seguinte é mais flexível. (Apesar de não ser o melhor modo, é um dos mais fáceis de se seguir.) Este método é chamado Método das Bolhas.

Vamos considerar uma lista de números a ser classificada:

7, 4, 5, 10, 2, 8

Nós queremos classificar esses números em ordem decrescente.

Vamos comparar os dois primeiros números. Se o número do lado esquerdo for menor que o do lado direito, então trocamos esses números entre si e comparamos o segundo e o terceiro números. Se os dois primeiros números já estiverem na ordem correta, devemos deixá-los como estão e comparar o segundo e terceiro números entre si.

Desse modo, no nosso exemplo, primeiro comparamos o 7 com o 4. Esses dois estão na ordem correta, de modo que vamos comparar o 4 com o 5. 4 é menor que 5, então trocamos os dois de lugar entre si. Assim, temos:

7, 5, 4, 10, 4, 2, 8

Agora comparamos o terceiro número com o quarto, isto é, 4 com 10. Trocamos outra vez os dois de lugar entre si:

7, 5, 10, 4, 2, 8

Comparando o quarto com o quinto, eles ficam onde estão. Finalmente, comparando o quinto número, 2, com o último, 8, nós os trocamos de lugar entre si, terminando com:

7, 5, 10, 4, 8, 2

Bem, ainda não terminamos! Mas observe que o número menor já está no final da lista. Vamos ver o que acontece quando passamos a segunda vez:

COMPARAR	TROCAR?	RESULTADO
prim./seg.	não	7, 5, 10, 4, 8, 2
seg./ter.	sim	7, 10, 5, 4, 8, 2
ter./quar.	não	7, 10, 5, 4, 8, 2
quar./quin.	sim	7, 10, 5, 8, 4, 2
qui./sex.	não	7, 10, 5, 8, 4, 2

No final da segunda passada, os dois menores números estarão em ordem no final da lista.

O número máximo de passadas que serão necessárias para ter todos os números em ordem, de qualquer lista é, dessa forma, um a menos que o número total de itens a serem classificados, isto é, no exemplo dado, o número máximo de passadas requerido será cinco. De fato, freqüentemente, as listas são classificadas bem antes, pois os itens estão quase na ordem correta antes de você iniciar a classificação. Dessa forma, é aconselhável verificar se a lista já está classificada no final de cada passada.

Vamos classificar agora a agenda telefônica em ordem alfabética. A primeira parte lê os dados em uma matriz; a segunda parte classifica os dados em ordem alfabética e a terceira parte exibe os dados classificados.

```
10 REM LISTA TELEFONICA ORDENADA
20 REM LE OS DATA E GUARDA NAS MATRIZES
30 E=4
40 N=E-1
50 DIM N$(N),NUM(N)
60 FOR I=0 TO N
70 READ N$(I),NUM(I)
80 NEXT I
90 REM ORDENACAO
100 M=N-1
110 P=0
120 FOR C=0 TO M
130 IF N$(C)>N$(C+1) THEN SWAP N$(C),N$(C+1): SWAP NUM
(C),NUM(C+1): P=P+1
140 NEXT C
150 IF P<>0 THEN GOTO 110
160 REM MOSTRA OS DADOS ORDENADOS
170 CLS
180 PRINT TAB(2)"NOME"TAB(19)"TELEFONE"
190 PRINT "****************************"
200 FOR I=0 TO N
210 LOCATE 2,4+I*2
220 PRINT N$(I)
230 LOCATE 18,4+I*2
240 PRINT NUM(I)
250 NEXT I
260 DATA EDUARDO,2677668,JOANA,4326843
270 DATA ANDRE',4781154,CARLOS,4781292
```

A primeira e a última parte deste programa deve ser familiar a você (veja o Capítulo 12).

As linhas 90 a 150 contêm a rotina de classificação.

No início do loop, o contador, P, é zerado. A cada vez que o loop se completar, um elemento da matriz é comparado ao seguinte, N$(0) é comparado a N$(1). Se os nomes desses elementos não estiverem em ordem alfabética, os conteúdos desses elementos são trocados entre si. P é agora fixado em 1, para indicar que uma troca já foi feita.

Na última passada do loop N$(2) é comparado a N$(3). Os conteúdos desses elementos são trocados se necessário e o contador P, incrementado.

A execução então continua na linha 150. Aqui o computador verifica o valor de P. Se qualquer um dos elementos foi trocado de lugar, P será maior que 0. O

computador assume que os nomes ainda não estão classificados e volta para a linha 110. P é zerado e o loop de classificação é executado outra vez.

Se, na linha 150, P estiver igual a 0, o computador sabe que nenhum elemento foi trocado de lugar, indicando, dessa forma, que os nomes estão agora em ordem alfabética. Por isso, o computador continua com a linha seguinte do programa e exibe a agenda.

Se você executar este programa, irá ver:

```
NOME          NÚMERO
********************************
ANDRÉ         4781154
CARLOS        4781292
EDUARDO       2677668
JOANA         4326843
```

Se você quiser usar o mesmo nome de matriz duas vezes em um programa, mas com dados diferentes, apague a matriz antiga:

```
ERASE N$
```

elimina todos os nomes armazenados na matriz N$. Agora você pode usar a matriz N$ outra vez, e redimensioná-la usando um comando DIM.

Se, por algum motivo, você tiver uma variável chamada N$, ela não será afetada pelo comando ERASE. Entretanto, procure não usar variáveis e matrizes com o mesmo nome, pois pode causar confusão.

Você pode ter matrizes multidimensionais com até 255 elementos! Por enquanto, considere uma matriz com apenas duas dimensões. Elas são muito úteis na classificação de tabelas:

DATA	CRÉDITO	DÉBITO	SALDO
30.06.84	34,05	10,00	34,05
05.07.84		10,00	24,05
10.07.84	120,00		144,05
12.07.84		50,00	94,05
20.07.84	200,00		294,05

Essas quatro colunas de números podem ser armazenadas na matriz bidimensional, BANCO(4,3). Essa matriz é grande o suficiente para armazenar uma tabela composta por cinco linhas e quatro colunas e pode ser representada conforme o seguinte:

BANCO(0,0)	BANCO(0,1)	BANCO(0,2)	BANCO(0,3)
BANCO(1,0)	BANCO(1,1)	BANCO(1,2)	BANCO(1,3)
BANCO(2,0)	BANCO(2,1)	BANCO(2,2)	BANCO(2,3)
BANCO(3,0)	BANCO(3,1)	BANCO(3,2)	BANCO(3,3)
BANCO(4,0)	BANCO(4,1)	BANCO(4,2)	BANCO(4,3)

Como essa é uma matriz numérica, a data será armazenada como um número, por exemplo, 300684. Cada entrada da tabela é armazenada em um elemento separado da matriz. A data 300684 pode ser armazenada em BANCO(0,0) e o saldo final em BANCO(4,3).

O programa seguinte mostra como colocar dados em uma matriz bidimensional. As quatro colunas da matriz são acessadas usando-se os valores de C = 0 a 3. O número de linhas depende do número de transações entradas; se houver T transações, então R assume o valor de 0 a T–1. Você insere a data de cada transação e a quantia creditada ou debitada, e o saldo de cada transação é calculado e inserido na coluna quatro da matriz. Os totais de créditos, débitos e saldo são encontrados e inseridos na última linha da matriz.

O cabeçalho da tabela é mantido em uma matriz de strings separada. Essa matriz não precisa ser dimensionada antes do uso, já que ela é uma matriz unidimensional e tem menos que onze elementos. Todas as matrizes multidimensionais devem ser DIMensionalizadas antes do uso.

```
10 REM BALANCETE BANCARIO
15 REM LE OPERACOES DOS DATA
20 CLS
30 FOR C=0 TO 3
40 READ OPER$(C)
50 NEXT C
60 REM ENTRADA DAS TRANSACOES
65 REM NUMA MATRIZ BIDIMENSIONAL
70 INPUT "NUMERO DE TRANSACOES";T
80 DIM BANCO (T,3)
90 N=T-1
100 CLS
110 FOR C=0 TO 2
```

```
120 PRINT TAB(10*C) OPER$(C);
130 NEXT C
140 FOR R=0 TO N
150 FOR C=0 TO 2
160 LOCATE 10*C,R+2
170 INPUT BANCO(R,C)
180 NEXT C
190 NEXT R
200 REM CALCULO DO BALANCETE DEPOIS DAS TRANSACOES
210 BANCO(0,3) = BANCO(0,1)-BANCO(0,2)
220 FOR R=1 TO N
230 X=BANCO(R,1)-BANCO(R,2)
240 BANCO(R,3)=BANCO(R-1,3)+X
250 NEXT R
260 REM CALCULO DO BALANCETE FINAL
270 FOR R=0 TO N
280 BANCO(T,1)=BANCO(R,1)+BANCO(T,1)
290 BANCO(T,2)=BANCO(R,2)+BANCO(T,2)
300 NEXT R
310 BANCO(T,3)=BANCO(R,1)-BANCO(T,2)
320 BANCO(T,0)=BANCO(N,0)
330 REM MOSTRA O BALANCETE DO BANCO
340 SCREEN 0
350 WIDTH 40
360 FOR C=0 TO 3
370 PRINT TAB(10*C) OPER$(C);
380 NEXT C
390 FOR R=0 TO N
400 FOR C=0 TO 3
410 LOCATE 10*C,R+2
420 PRINT BANCO(R,C)
430 NEXT C
440 NEXT R
450 PRINT "-------------------------------------"
460 FOR C=0 TO 3
470 PRINT TAB(10*C) BANCO(T,C);
480 NEXT C
490 END
500 DATA FICHA,CREDITO,DEBITO,BALANCETE
```

Se você inserir débitos ou créditos maiores que 999999.99, irá "bagunçar" a tabela, já que ela tem somente dez espaços de caracteres alocados para cada inserção.

EXERCÍCIO

1. Altere ligeiramente este programa de modo que você possa expressar a data na forma 23.06.84 etc. Você irá precisar usar outra matriz de string; DIA$(T) será aceitável.

CAPÍTULO 14

MANIPULANDO STRINGS

Existem muitas maneiras pelas quais você pode manipular strings. Até agora você só somou uma string a outra:

```
PRINT "Bom "+" dia"
```

é impressa como:

```
Bom dia
```

Usando INSTR você pode procurar uma string específica dentro de uma string. Em um programa para um adventure*, você pode querer perguntar ao jogador para onde ele irá a seguir. O jogo pode seguir diferentes caminhos dependendo se o jogador digitar Norte, Sul, Leste ou Oeste. Esta listagem de parte de um programa mostra como isso pode ser feito:

```
100 INPUT "PARA ONDE VAI AGORA?";B$
110 IF B$="NORTE" GOTO 200
120 IF B$="SUL" GOTO 300
130 IF B$="LESTE" GOTO 400
140 IF B$="OESTE" GOTO 500
150 PRINT "TENTE NOVAMENTE!": GOTO 100
```

* Tipo de jogo no qual os participantes interagem com o micro através de textos.

Entretanto, o jogador pode muito bem ter digitado uma sentença completa em resposta ao INPUT da linha 100, por exemplo, EU ESTOU INDO PARA LESTE.

No entendimento do programa, isto significaria que nenhum dos caminhos poderá ser executado e o jogador será perguntado outra vez para onde ele está indo agora. Isso deve ser repetido até que uma palavra simples como "LESTE" seja inserida, fazendo com que o jogo demore muito mais. Muito melhor seria se o computador pudesse pesquisar a string inteira para reconhecer uma palavra dentro da string. Isso é exatamente o que a função INSTR faz. O exemplo seguinte irá mostrar como:

```
100 INPUT "PARA ONDE VAI AGORA?";B$
110 IF INSTR(B$,"NORTE")<>0 GOTO 200
120 IF INSTR(B$,"SUL")<>0 GOTO 300
130 IF INSTR(B$,"LESTE")<>0 GOTO 400
140 IF INSTR(B$,"OESTE")<>0 GOTO 500
150 PRINT "TENTE NOVAMENTE!": GOTO 100
```

Na linha 110, a função INSTR pesquisa a string principal, B$, procurando a string "NORTE". Se a encontrar, é retornado o número correspondente à posição da primeira letra de "NORTE", na string B$. Se LESTE não for encontrado, INSTR retorna a zero. Desse modo, se B$ for:

```
"EU ESTOU INDO PARA O NORTE AGORA"
```

então, INSTR(B$,"NORTE") retorna o número 22, já que a letra N de NORTE ocorre na vigésima segunda posição de caractere. Para verificar, digite este pequeno programa:

```
10 A=INSTR("EU ESTOU INDO PARA O NORTE","NORTE")
20 PRINT A
```

O número 22 deve ser impresso. INSTR sempre retorna um número, de modo que deve sempre ser comparada com variáveis numéricas.

Seja cuidadoso ao incluir strings na função INSTR, dentro das aspas. Variáveis string já incluem aspas, sendo que elas devem ser colocadas de modo a permanecerem dentro da função INSTR sem aspas.

Quando estiver pesquisando um string, você pode iniciar de qualquer ponto especificado:

```
INSTR(10,A$,B$)
```

Isso faz com que se inicie a pesquisa de BS em A$, a partir da posição do décimo caractere.

Esse comando pode ser útil em um programa de Forca. O programa seguinte pesquisa a palavra:

CONSTANTINOPLA

procurando a letra N

```
10 A$="CONSTANTINOPLA"
20 B$="N"
30 C=0
40 C=INSTR(C+1,A$,B$)
50 IF C=0 THEN END
60 PRINT "EXISTE UM ";B$;" NA POSICAO"C
70 GOTO 40
```

Neste programa, a expressão C+1 foi usada para indicar onde a pesquisa deve começar. Na primeira vez que a função INSTR encontrar um "N", C = 3, a pesquisa seguinte começa a partir da posição 4. Após ter sido encontrado o terceiro "N", C = 10 e a terceira e última pesquisa começa na posição de caractere 11. Não existem mais "N"s, de modo que C é zerado fazendo com que o programa termine na linha 50.

Você receberá sempre um zero como retorno da função INSTR se tentar pesquisar uma string maior dentro de uma menor.

INSTR("dia","bom-dia")

irá retornar um zero.

Agora vamos ver três funções muito similares:

RIGHT$, LEFT$ e MID$

Em primeiro lugar RIGHT$. Eis um exemplo de seu uso:

```
10 A$=RIGHT$("QUE TEMPO FEIO ESTA LA FORA",4)
20 PRINT A$
```

fornece:

```
FORA
```

Os cinco caracteres mais à direita foram retornados na variável string A$.

Espero que você possa adivinhar o que LEFT$ faz! De qualquer modo, eis um exemplo:

```
10 A$=LEFT$("QUE TEMPO FEIO ESTA LA FORA",10)
20 PRINT A$
```

fornece:

```
QUE TEMPO
```

Finalmente MID$:

```
10 A$=MID$("QUE TEMPO FEIO ESTA LA FORA",11,17)
20 PRINT A$
```

fornece:

```
FEIO ESTÁ LA FORA
```

O primeiro número na função MID$ fornece a posição de caractere para o primeiro caractere a ser retornado. O segundo número diz ao computador quantos caracteres devem ser retornados. Desse modo, no exemplo acima, os quatorze caracteres a partir da nona posição foram retornados dentro da variável A$.

O número usado nessas funções deve estar entre 1 e 255. Como de qualquer forma você não pode ter uma linha maior que 255 caracteres, isso faz sentido.

Se o primeiro argumento numérico em MID$ for maior que o número de caracteres da string, então você irá receber como retorno uma string vazia.

Mais uma vez, não se esqueça das aspas e não tente usar variáveis numéricas com essas três funções; você deve usar variáveis string já que são retornadas strings por essas funções.

EXERCÍCIOS

1. Usando INSTR, encontre os resultados de:

a) Pesquisa de uma string vazia ou nula, isto é, "".

b) Pesquisa de uma string dentro de uma string vazia.

c) Pesquisa de uma string vazia dentro de uma string vazia.

2. Imprima as palavras existentes em "Tem alguém aí?" separadamente. Você pode usar INSTR para descobrir a posição de cada palavra e só então reutilizar MID$, ou usar RIGHT$, LEFT$ e MID$.

3. Altere o último programa de modo que você possa separar as palavras de qualquer string que você inserir.

CAPÍTULO 15

FUNÇÕES

Você já viu algumas funções, como por exemplo:

INT	Capítulo 4
INSTR	Capítulo 14
LEFT$	Capítulo 14
MID$	Capítulo 14

Falando de modo geral, cada uma delas age da mesma forma. Você tem um objeto, chamado argumento, no qual a função age para produzir um resultado. Uma vez definida uma função, ela sempre produz os mesmos resultados e do mesmo modo se for fornecido um argumento aceitável:

```
INT(X)
```

Aqui, o argumento é X. O argumento de uma função é sempre incluído entre parênteses. O argumento, neste caso, pode ser um número, uma variável numérica ou uma expressão numérica. Se X estiver nos limites, INT sempre irá retornar o valor inteiro de X arredondando para o valor inteiro inferior mais próximo.

INT(7.9)	Retorna 7.
INT(–2.3)	Retorna –3.

Se você ficar confuso com certos números negativos, veja os números da linha abaixo:

...−6 −5 −4 −3 −2 −1 0 1 2 3 4 5...

Os números estão arranjados na ordem crescente de tamanho, indo da esquerda para a direita. Com isso, você pode ver que −3 é o inteiro menor mais próximo de −2.3.

Existe uma outra função muito similar a INT, chamada FIX.

FIX também age sobre um argumento de número real para retornar um inteiro. No caso de números positivos, ela retorna o inteiro menor mais próximo da mesma forma que INT, mas para números negativos, ela retorna o inteiro maior mais próximo, ao contrário de INT:

```
FIX(5.5)     Retorna 5.
FIX(−2.2)    Retorna −2.
```

Vamos ver mais duas funções que também têm argumentos numéricos: ABS e SGN.

SGN retorna o sinal do argumento:

- Se o argumento for um número negativo, a função retorna −1.
- Se o argumento for 0, a função retorna 0.
- Se o argumento for um número positivo, a função retorna +1.

```
10    A=SGN(−402.2)
20    PRINT "SGN(−402.2)=";A
```

fornece:

```
SGN(−402.2)=−1
```

Por outro lado, ABS é usada para converter números negativos em positivos. Números positivos não são alterados por esta função:

```
10 A=ABS(-402.2)
20 PRINT "ABS(-402.2)=";A
```

fornece:

```
ABS(-402.2)=402.2
```

O argumento de ambas as funções pode ser ou expressões ou variáveis:

SGN(4+3*-2)	Retorna -1.
ABS(num)	Converte o conteúdo da variável num em um número positivo.

Eis um pequeno programa para mostrar um uso possível para ABS:

```
10 REM ESTE PROGRAMA CALCULA A SUA IDADE
20 INPUT "DATA DE HOJE: MES,ANO";MH,AH
30 INPUT "DATA DO SEU NASCIMENTO: MES,ANO";MN,AN
40 MI=ABS(MH-MN): IF MN>MH THEN MI=12-MI
50 AI=ABS(AH-AN): IF MN>MH THEN AI=AI-1
60 PRINT "VOCE TEM";AI;" ANOS E";MI" MESES"
```

Agora vamos ver duas funções que trabalham tanto com números como com strings: VAL e STR$.

Se uma string contiver um número, VAL irá retornar o número dentro de uma variável numérica, isto é, ela elimina as aspas e remove quaisquer brancos de dentro da string. Por exemplo:

```
A=VAL("  273")
PRINT  "A=";A
```

fornece:

```
A = 273
```

Se existir mais de um número dentro de uma string, VAL somente retorna o primeiro número e vai em frente. Por essa razão, VAL não avalia expressões dentro de strings:

```
PRINT VAL("3+4=7")
```

fornece:

```
3
```

Não se esqueça de que, apesar do fato do argumento de VAL ser uma string, o resultado retornado por VAL é um número e deve ser colocado em uma variável numérica:

```
A$=VAL(7)          está errado
```

VAL não consegue trabalhar com números embutidos entre letras em uma string:

```
10 A=VAL("ESTOU 1ANO MAIS VELHO")
20 PRINT A
```

fornece:

```
0
```

A função inversa à VAL é STR$; ela converte um argumento numérico em uma string:

```
A$="Eu tenho "+STR$(60)+" anos hoje"
PRINT A$
```

fornece:

```
Eu tenho 60 anos hoje
```

Não se esqueça de que, apesar do argumento ser um número, o resultado é uma string, de modo que você deve retorná-lo em uma variável string.

No Capítulo 13 usamos uma Matriz Numérica, BANCO, para armazenar datas, créditos, débitos e saldos. Infelizmente, a data teve de ser inserida como um número, isto é, 240684 em vez de 24/06/84. Para inseri-la da segunda forma, você tem de armazenar as datas separadamente em outra matriz de string de modo que o programa use uma matriz numérica para as transações e duas matrizes de string para cabeçalhos e datas. Usando STR$, todos os dados podem ser armazenados em uma matriz de string grande. Para isso, têm de ser feitas as seguintes alterações:

```
170 INPUT  BANCO
180 BANCO$(R,C)=STR$(BANCO)
```

Para calcular o novo saldo após cada transação, você deve avaliar as strings nas colunas dos débitos e créditos usando VAL e então converter a resposta de volta às strings:

```
210 BANCO$(0,3)=STR$(VAL(BANCO$(0,1))-VAL(BANCO$(0,2))
)
```

Talvez você queira voltar atrás e alterar o programa de SALDO BANCÁRIO!

Uma última função que atua sobre strings é LEN. LEN retorna o comprimento da string fornecida no argumento:

```
10 A$="CONSTANTINOPLA"
20 A=LEN(A$)
30 PRINT "O COMPRIMENTO DA PALAVRA "A$" E' DE "A" CARA
CTERES"
```

Outra vez, observe que, apesar da função atuar sobre uma string, ela sempre retorna um número, de modo que você deve comparar essa função com variáveis numéricas.

Finalmente, uma função um pouco diferente: RND. RND gera um número aleatório de 14 dígitos entre 0 e 1. De fato, os números gerados não são realmente aleatórios já que seguem uma string fixa. Entretanto, essa string é tão longa que, para todos os propósitos, você pode considerar que os números gerados são realmente aleatórios.

Você pode ter três tipos de argumentos com RND:

Se o argumento for positivo, o número gerado por RND dentro de um programa é o seguinte na string. O ponto no qual a string é inserida é sempre o mesmo, e ele é estabelecido pelo computador no final do programa. Digite:

```
10 PRINT RND(1)
20 PRINT RND(1)
30 PRINT RND(1)
```

Você irá obter a seguinte string a cada vez que o programa for executado:

```
.59521943994623
.10658628050158
.76577651772823
```

Alterar o argumento para um número positivo diferente não irá produzir qualquer efeito.

Se o argumento for zero, o número aleatório gerado é o mesmo que o último. Por exemplo:

```
10 X=RND(1)+RND(1)
20 PRINT " O SEGUNDO NUMERO ALEATORIO GERADO E'"RND(0)

30 PRINT " O PRIMEIRO NUMERO ALEATORIIO GERADO E'"X-RN
D(0)
```

Se o argumento for negativo, o lugar na string na qual os números aleatórios são detectados é definida pelo argumento negativo:

```
10 A=RND(-3)
20 B=RND(1)
30 PRINT A,B
```

fornece a string:

```
.84389820420821
.29624868166920
```

a cada vez que o programa for executado. Agora substitua –3 por –4. Você irá obter uma string diferente. Alterando o argumento positivo, não irá produzir outra vez qualquer resultado.

Um número aleatório entre 0 e 1 não tem muita utilidade. É preferível ter um número aleatório entre 1 e 6. Por exemplo, este programa simula lançamentos de dados:

```
10 REM AREMESSO DE DADOS
20 A=INT(6*RND(1)+1)
30 PRINT "SAIU O NUMERO"A
```

Entretanto, você irá obter o mesmo número a cada vez que o dado for lançado. Se você quiser obter um número aleatório diferente a cada lançamento, coloque a linha:

```
5 RDRST=RND(-TIME)
```

no início do programa.

A função TIME retorna o valor do relógio interno do computador. Ao se ligar a máquina, o relógio é posicionado em 0, mas é incrementado a cada 1/60 segundos, isto é, TIME será igual a 60 quando tiver passado um segundo. Desse modo, usando TIME como argumento de uma função RND, irá resultar em uma string de números aleatórios sendo lidos em pontos diferentes a cada vez que a função RND for encontrada no programa.

Você pode ver como usar esta última função no jogo da Forca? O programa pode ter este tipo de estrutura:

1. Carregue uma matriz com as palavras contidas em comandos DATA.
2. Escolha aleatoriamente um dos elementos (palavra) da matriz.
3. Uma vez lida a sua palavra, informe o seu comprimento ao jogador, usando LEN.
4. Peça ao jogador para adivinhar uma letra.
5. Usando INSTR, verifique se a letra existe na palavra (veja o Capítulo 14, caso você tenha se esquecido como fazer isso).
6. Conte as tentativas que o jogador fez.
7. Se o jogador não conseguir adivinhar a palavra em dez tentativas, termine o programa. O computador venceu.
8. Pergunte ao jogador se ele deseja jogar outra vez, usando INKEY$.

EXERCÍCIO

1. Escreva o programa do jogo da Forca.

CAPÍTULO 16

DEFININDO AS SUAS PRÓPRIAS FUNÇÕES

Vamos ver algumas funções definidas por nós mesmos, deixando um pouco de lado as definidas pelo computador. Isso pode ser feito usando DEF FN.

Uma vez definida uma função, você pode usá-la em qualquer parte do programa.

Em primeiro lugar, vamos ver um exemplo matemático.

Suponha que você queira definir uma função que calcule o quadrado de um número, isto é, eleve esse número à potência de dois, que é a mesma coisa que multiplicá-lo por si próprio.

Vamos chamar a essa função QUADRADO e defini-la como:

```
10  DEF FNQUADRADO(X)=X^2
```

Observe que o nome da função vem logo após o comando DEF FN.

X é chamado "parâmetro fictício". Você somente o usa para mostrar o que a função faz. Quando usar essa função, você deve substituir o parâmetro fictício pelo parâmetro real. O parâmetro real tem de ser do mesmo tipo do parâmetro fictício. No exemplo dado, o parâmetro real pode ser ou uma variável numérica ou um número. Quando definir a função, o parâmetro fictício usado na definição deve ser listado dentro de parênteses após o nome da função.

Vamos usar essa função para calcular o quadrado de 13 e colocá-lo na variável R. Para isso, usamos o comando FN, seguido pelo nome da função e então pelo parâmetro real entre parênteses:

```
R=FNQUADRADO(13)
```

Ao examiná-lo, o computador irá procurar a definição da função QUADRADO. Ao encontrá-la, toda a definição de X é substituída por 13, o resultado é calculado e colocado na variável R.

X é também chamado variável local. Isto significa que ela é somente reconhecida localmente, isto é, dentro da função:

```
10 X=4
20 DEF FNQUADRADO(X)=X^2
30 R=FNQUADRADO(12)
40 PRINT "R="R"X="X
```

fornece:

```
R=144  X=4
```

Você pode usar mais de um parâmetro em uma função. Todos os parâmetros usados devem ser listados na definição, com os parâmetros separados por vírgulas. Quando chamar a função, você deve passar o mesmo número de parâmetros reais da função que os parâmetros fictícios usados na definição. Você também deve passar o tipo correto de parâmetro, isto é, um número ou variável numérica para um parâmetro numérico fictício.

O programa seguinte irá elevar qualquer número à potência de outro número, sendo ambos os números passados na definição da função:

```
20 DEF FNELEVA(X,Y)=X^Y
30 INPUT "BASE ";A
40 INPUT "EXPOENTE ";B
50 R=FNELEVE(A,B)
60 PRINT A"^"B"="R
```

A definição deve estar contida em uma linha de programa.

Agora vamos ver um exemplo usando strings. Essa função troca o primeiro caractere de uma função:

```
10 DEF FNST$(A$)=RIGHT$(A$,LEN(A$)-1)
20 INPUT "STRING=";S$
30 PRINT S$,FNST$(S$)
```

Como essa função retorna uma string, o nome da função deve terminar com o sinal $. Qualquer nome de variável válido é aceitável como nome de função.

EXERCÍCIO

1. Defina uma função que corte os últimos quatro caracteres de uma string.

CAPÍTULO 17

ESTRUTURANDO O SEU PROGRAMA

Em um programa, freqüentemente você pode se encontrar em situações em que deseja executar a mesma tarefa muitas vezes, em diferentes pontos de seu programa. Se você descobrir que é necessário repetir uma linha de programa muitas vezes, é aconselhável colocar essa linha no que é chamado "sub-rotina". Quando você precisar dessa linha em seu programa, você salta para a sub-rotina, usando o comando GOSUB. Quando todas as linhas da sub-rotina tiverem sido executadas, você retorna ao programa principal. Uma sub-rotina pode conter tantas linhas de programa quantas necessárias. No exemplo seguinte, a sub-rotina está entre as linhas 100 e 130:

```
10 CLS
20 PRINT "VOCE GOSTA DE MIM?"
30 GOSUB 110
40 PRINT "VOCE ME ACHA INTELIGENTE?"
50 GOSUB 110
60 PRINT "VOCE ME ACHA BONITO?"
70 GOSUB 110
80 PRINT "VOCE SO ESTA DIZENDO ISSO PARA ME AGRA-DAR!"
90 END
100 REM SUB-ROTINA
110 INPUT "SIM ou NAO ";SN$
120 IF SN$="NAO" OR SN$="nao" THEN PRINT "OK.ENTAO EU
VOU EMBORA.":END
130 RETURN
```

Observe o comando RETURN no final do programa. Isso é muito importante, pois ele diz ao computador para retornar ao programa principal. A execução continua a partir da primeira linha após a linha que contém o comando GOSUB.

É conveniente colocar um comando REM exatamente antes da sub-rotina para distingui-la do programa principal. Entretanto, não use o número de linha do comando REM para chamar a sub-rotina, pois, se mais tarde você vier a apagar os comandos REM, irá receber uma mensagem de erro.

Assegure-se de que o computador não execute a sub-rotina por acidente, colocando um comando END no final do programa principal, exatamente antes das sub-rotinas.

Outro uso para o comando GOSUB é com o comando IF...THEN...ELSE. Esse comando tende a se tornar longo e complicado; a maneira mais fácil de simplificar esse comando é usando uma sub-rotina.

O pequeno programa a seguir pede dois números positivos diferentes. O comando IF...THEN...ELSE é usado para testar esses números. Se eles forem diferentes e positivos, o seu produto e soma são impressos. De outra forma, você será solicitado a tentar outra vez:

```
10 INPUT "DOIS NUMEROS POSITIVOS DIFERENTES";A,B
20 IF A=B OR A<0 OR B<0 THEN PRINT "TENTE DE NOVO!" EL
SE GOSUB 100
30 GOTO 10
90 REM SUB-ROTINA
100 PRINT "A=";A,"B=";B
110 PRINT "A+B=";A+B
120 PRINT "AxB=";A*B
130 RETURN
```

O programa é ligeiramente mais longo, mas muito mais fácil de ser lido.

CAPÍTULO 18

DESVIOS CONDICIONAIS

Em determinado ponto você pode descobrir que deseja ter um programa principal ou programa de menus, que ofereça um número de opções. Por exemplo, as primeiras linhas de um programa de menu podem fazer com que as seguintes linhas sejam exibidas na tela:

```
**************** MENU ****************
 1. Exibe a agenda telefônica.
 2. Inclui novas entradas na agenda.
 3. Remove entradas da agenda.
 4. Termina o programa.
**************************************
```

A parte seguinte do programa pode se parecer com esta:

```
100  A$=INKEY$
120  IF A$="1" THEN GOSUB 1000
130  IF A$="2" THEN GOSUB 2000
140  IF A$="3" THEN GOSUB 3000
150  IF A$="4" THEN GOSUB 4000
160  GOTO 100
```

A linha 160 foi incluída para o caso de outra tecla diferente de 1, 2, 3 ou 4 ser pressionada.

Existe uma maneira mais curta de escrever um programa, usando ON GOSUB. Usando este comando, você pode saltar para qualquer número de sub-rotina que quiser. No exemplo anterior, queremos saltar para uma das quatro sub-rotinas após a palavra GOSUB:

```
ON...GOSUB   1000,2000,3000,4000
```

Agora temos de fazer o computador saber para qual sub-rotina saltar. Isto é feito colocando um número ou expressão após ON e antes de GOSUB. Se a expressão for igual a 1, o computador salta para a primeira sub-rotina listada após GOSUB; se for igual a 2, para a segunda sub-rotina, e assim por diante.

Agora podemos substituir as linhas 100 a 160 por:

```
100  A$=INKEYS
110  A=VAL(A$)
120  ON A GOSUB 1000,2000,3000,4000
130  GOTO 100
```

Observe que a string A$ foi alterada para a variável numérica A, antes de usá-la no comando ON...GOSUB. De outra forma, iria causar um erro, pois somente variáveis numéricas podem seguir ao comando ON.

Ao terminar a sub-rotina, o computador retorna para a linha 130.

Se a variável numérica A contiver um número real, por exemplo, 2.9, o computador pegará o valor inteiro menor e saltará para a segunda sub-rotina.

ON...GOTO trabalha da mesma forma que ON...GOSUB. Neste caso, o computador salta para o número de linha na lista que corresponde ao número que segue ao comando ON. A execução do programa continua a partir desse número de linha:

```
10  INPUT "1, 2 ou 3";A
20  ON A GOTO 370,420,10
```

Desse modo, se:

```
A=1    A linha seguinte a ser executada é a 370.
A=2    A linha seguinte a ser executada é a 420.
A=3    A linha seguinte a ser executada é a  10.
```

Observe que os números de linha não precisam estar na ordem numérica na lista.

CAPÍTULO 19

FUNÇÕES MATEMÁTICAS

Se você não tiver muita experiência com funções matemáticas e achar que este capítulo é um pouco difícil, não hesite em pular para o seguinte.

Este capítulo trata das funções matemáticas:

^, SQR
TAN, SIN, COS e ATN
EXP e LOG

Você já sabe como elevar um número a uma potência usando o sinal:

4^2 = 16 (ou escrito na forma usual $4^2=16$)

Geralmente, a elevação de um número n, a uma potência p, é o equivalente a multiplicar n por ele próprio, p vezes. No exemplo anterior, n = 4, p = 2. Desse modo 4^2 é equivalente a 4*4.

A função raiz quadrada, SQR, encontra a raiz quadrada de um número. Isso é o inverso de elevar um número à potência de dois, ou elevá-lo ao quadrado.

Se você elevar 5 ao quadrado, isto é, 5^2, você obtém 25. Para tirar 5 de volta a 25, você usa a função raiz quadrada:

```
10 A=SQR(25)
20 PRINT "A raiz quadrada de 25 e";A
```

obtendo:

```
A raiz quadrada de 25 é 5
```

O argumento da função SQR deve ser um número positivo ou uma expressão que resulte um número positivo. (O computador não pode manusear números imaginários!) Eis alguns pontos para se pensar a respeito: o que acontece se você elevar um número a uma potência negativa? É fácil descobrir. Digite:

```
PRINT 4^-1
```

Você irá obter:

```
.25
```

que é igual a 1/4, de modo que, geralmente:

```
n^-p = 1/n^p
```

Vamos ver o que acontece se um número for elevado à potência de um número decimal, ou fração. Digite:

```
PRINT 4^0.5
```

fornece a resposta 2.

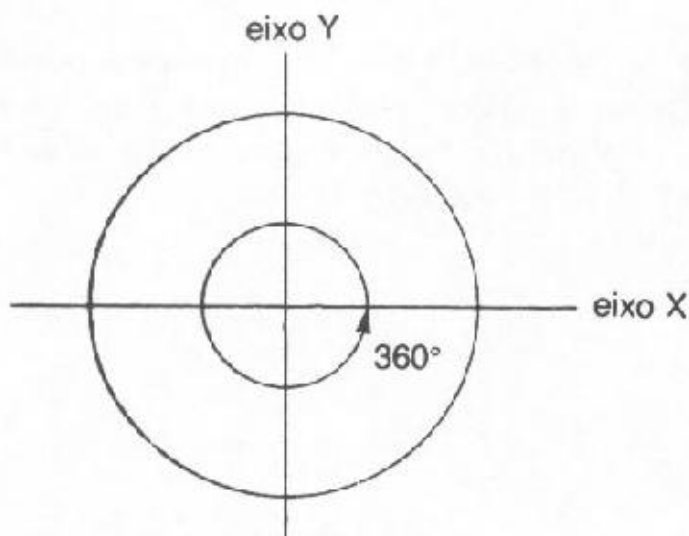
4^0.5 é o equivalente a $4^{1/2}$

de modo que, geralmente:

```
n^1/p fornece a você a raiz p de n
```

No exemplo, n = 4, p = 2, você obtém a raiz quadrada de 4, que é 2. Desse modo, elevar um número à potência de meio é o equivalente a se obter a raiz quadrada do número.

SIN, COS, TAN e ATN são funções trigonométricas.

Os argumentos dessas três funções devem ser ângulos. Provavelmente você pensa em ângulos em termos de graus:

Um círculo completo tem 360°. O ângulo de um círculo é medido a partir do eixo X, que corta o círculo com uma linha traçada a partir da posição das nove horas para a posição das três horas, no sentido anti-horário. O eixo Y corta o círculo a partir da posição das doze horas para a posição das seis horas. O ângulo entre esses dois eixos é, dessa forma, 90°.

Entretanto, o computador mede ângulos de forma ligeiramente diferente. Para entender, vamos considerar um círculo com um raio de uma unidade. O tipo de unidade não importa aqui; pode ser 1 mm, 1 m, 1 km, qualquer uma.

A circunferência do círculo é dada pela fórmula:

C=2*PI*r   C = Circunferência.
PI é um número especial usado em matemática, que inicia com 3,1415926...
PI é a letra grega "p" e é pronunciada PI.
r é o raio do círculo.

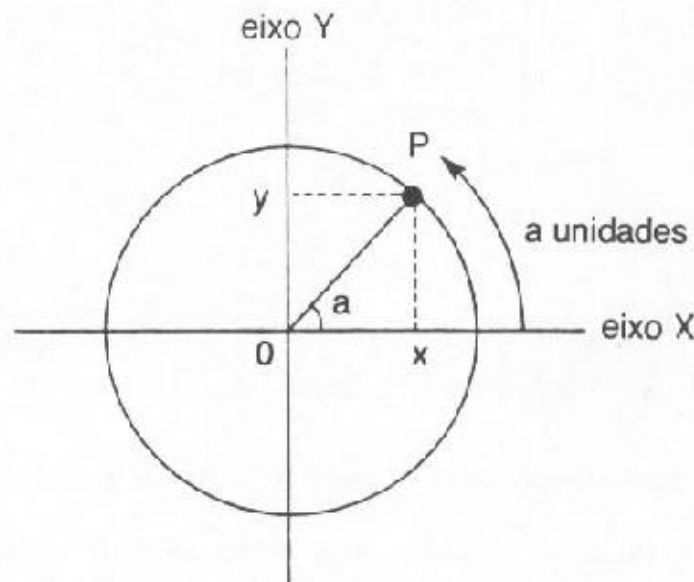
Dessa forma, para o nosso círculo, como r = 1:

C = 2*PI

Examinando por outro ângulo, podemos dizer que a circunferência de nosso círculo tem 2*PI radianos. Dessa forma:

2*PI radianos é equivalente a 360°.

Assim, em radianos, o ângulo entre o eixo X e o eixo Y é PI/2.

Se, agora, nós tivermos um ponto se movendo ao longo do nosso círculo, podemos usar as funções COS e SIN para descobrir onde o ponto está:

A um certo tempo, o ponto pode estar na posição P, para onde ele se moveu, através de um ângulo a partir do eixo X; ou, se você preferir, ele se moveu uma distância de "a" unidades ao longo da circunferência.

Se desenharmos linhas tracejadas verticais e horizontais a partir de P, para cortar os eixos X e Y, podemos descobrir a posição das intersecções nos eixos X e Y usando COS e SIN respectivamente:

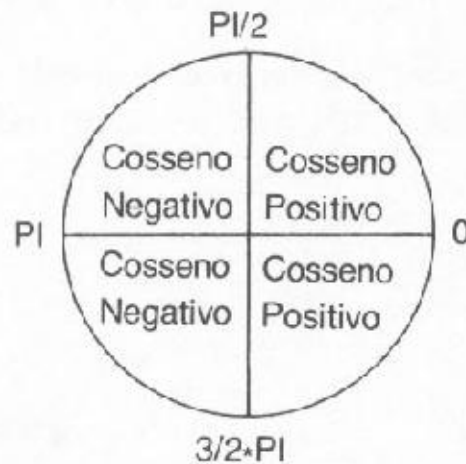
X = COS(a)   COS fornece o cosseno do ângulo a.
Y = SIN(a)   SIN fornece o seno do ângulo a.

Quando medimos ao longo do eixo X, consideramos X positivo se estivermos no lado direito do eixo Y e X negativo quando estivermos do lado esquerdo do eixo Y. Desse modo podemos dizer:

- O cosseno de qualquer ângulo entre 0 e PI/2 é positivo.

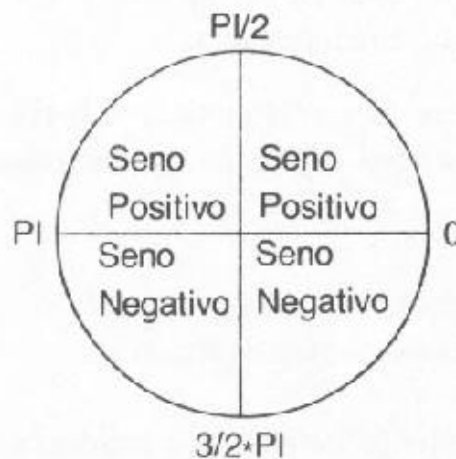
· O cosseno de qualquer ângulo entre PI/2 e 3*PI/2 é negativo.

· O cosseno de qualquer ângulo entre 3*PI/2 e 2PI é positivo.

Isto é, para o COS:

Quando medimos ao longo do eixo Y, Y é positivo se estiver acima do eixo X e negativo quando abaixo. Desse modo podemos dizer:

· O seno de qualquer ângulo entre 0 e PI é positivo.

· O seno de qualquer ângulo entre PI e 2*PI é negativo.

Isto é, para SIN:

Se quisermos andar ao longo do círculo mais de uma vez, COS e SIN têm os mesmos valores que da primeira vez:

```
SIN(a+2*PI)=SIN(a)
COS(a+2*PI)=COS(a)
```

A tangente de um ângulo é dada por:

```
TAN(a)=SIN(a)/COS(a)
```

Dada a tangente de um ângulo, podemos descobrir o ângulo usando ATN (que é conhecida por co-tangente). ATN é o inverso da função TAN.

```
a=ATN(0.6)
```

retorna o ângulo cuja tangente é 0.6 na variável a. O ângulo é retornado em radianos.

O computador não conhece as funções inversas de COS e SIN, mas você pode facilmente defini-las como suas próprias funções usando:

```
ARCSIN(a)=ATN(a/SQR(-a*a+1))
ARCCOS(a)=-ATN(a/SQR(-a*a+1))+PI/2
```

Outro número freqüentemente encontrado em matemática é representado pela letra e.

Para ver os primeiros 14 dígitos desse número, digite:

```
PRINT EXP(1)
```

Você irá ver:

```
2.7182818284588
```

A função EXP é definida por:

```
EXP(x)=e^x
```

de modo que:

```
EXP(1)=e
```

O inverso dessa função é dado pela função LOG. Essa função fornece o logaritmo na base e do argumento, isto é, ele fornece o logaritmo natural de um número que freqüentemente é escrito ln.

```
EXP(X)=e^x=n
LOG(n)=x
```

Você usará com mais freqüência logaritmos na base 10. Para se obter logaritmos na base 10, use a seguinte identidade:

```
log10(x)=LOG(x)/LOG(10)
```

EXERCÍCIOS

1. Defina uma função chamada PI, que retorna o valor de PI. Use a identidade:

```
PI=ATN(1*4)
```

2. Defina uma função que converta graus em radianos. Use a fórmula:

```
a radianos=(a*PI)/180 graus
```

Você terá de usar 3.14... para o PI. Alternativamente, use a função PI.

3. Defina uma função que forneça o ARCCOS de seu argumento. Não se esqueça, –1 < = COS(a) < = 1 para qualquer ângulo de "a" radianos. Desse modo, o valor de sua função somente funcionará para argumentos entre +1 e –1, inclusive.

4. Defina uma função que retorne o logaritmo na base 10 de seu argumento.

CAPÍTULO 20

O CÓDIGO ASCII

No Capítulo 6, você descobriu que as letras de A a Z e de a a z são armazenadas dentro do computador como números entre 65 e 90 e 97 e 122, respectivamente. Esses números são chamados Códigos ASCII.

ASCII significa *American Standard Code for Information Interchange* (Código Americano Padrão para Intercâmbio de Informações), e é comumente usado pelos computadores para representar caracteres.

Se você quiser descobrir os códigos ASCII para um caractere, você pode usar o comando ASC. Por exemplo:

```
10 A1=ASC("A"):A2=ASC("a")
20 PRINT "O codigo ASCII de A e";A1
30 PRINT "O codigo ASCII de a e";A2
```

O argumento da função ASC deve ser uma string; se mais de um caractere estiver contido na string, o computador retorna o código ASCII do primeiro caractere.

O oposto ao comando ASC é CHR$. Quando fornecido um argumento aceitável, CHR$ retorna o caractere ASCII correspondente, em uma string. Por exemplo:

```
10 A1$=CHR$(65):A2$=CHR$(97)
20 PRINT "O caractere correspondente ao codigo 65 do A
SCII e ";A1$
30 PRINT "O caractere correspondente ao codigo 97 do A
SCII e ";A2$
```

Os outros caracteres do teclado também têm códigos ASCII. Encontre os códigos ASCII para "%", "?" e "7".

De fato, existem 256 códigos ASCII ao todo, entre 0 e 255.

Você pode usar CHR$ para descobrir caracteres associados com os códigos entre 32 e 255 inclusive. Execute o seguinte programa para descobrir esses caracteres:

```
10 REM Caracteres ASCII :32-255
20 CLS
30 FOR I=32 TO 255
40 PRINT CHR$(I)+" ";
50 NEXT I
```

Você verá todos os caracteres alfanuméricos como também a maioria dos caracteres gráficos. Observe que o último caractere, que é representado pelo código 255, é de fato o cursor; dessa forma, ele muda à medida que você move o cursor sobre o texto da tela.

Os códigos entre 0 e 31 são um tanto especiais.

Esses códigos são chamados códigos de controle. Em vez de representar um caractere, eles representam uma operação, que pode ser executada pelo computador quando o código for usado como argumento em um comando PRINT CHR$.

Você já viu o código de controle 13 no Capítulo 8. Fazendo um PRINT CHR$ vimos que era o equivalente a pressionar <RETURN>. Essa é a única maneira como <RETURN> pode ser usado dentro de um programa.

Outros códigos de controle movem o cursor ao longo da tela, limpam a tela, etc.

Tente executar o seguinte programa:

```
10 REM alguns Codigos de Controle ASCII
20 PRINT CHR$(12)+"Codigo de Controle 12 limpa a tela.
"
30 PRINT CHR$(7)+"Codigo de Controle 7 emite BEEPs."
```

Os códigos de 0 a 31 são também usados para representar o restante dos caracteres gráficos. Entretanto, para acessá-los você deve usar PRINT CHR$(1) em primeiro lugar e então CHR$(A+64), onde A é o número do código. O programa seguinte imprime os 32 primeiros caracteres gráficos:

```
10 REM Codigos 0-31:mais Caracteres Graficos
20 FOR A=0 TO 31
30 PRINT CHR$(1)+CHR$(A+64)+" ";
40 NEXT A
```

A função STRING$ é usada para imprimir uma string de um dado tamanho e constituída por um ou outro caractere específico. A string não pode exceder a 200 caracteres, a menos que você use primeiro um comando CLEAR.

STRING$ deve ser seguida por dois argumentos, separados por uma vírgula e incluídos entre parênteses.

O primeiro argumento deve ser um número, expressão numérica ou variável numérica. Isso especifica o tamanho de uma string.

O segundo argumento pode ser ou um número ou uma string. Se um número for usado, pode ser o código ASCII do caractere a ser impresso na string. Por exemplo:

```
10 FOR B=1 TO 10
20 A$=STRING$(B,42)
30 PRINT A$
40 NEXT B
```

fornece o padrão:

```
*
**
***
****
*****
******
*******
********
*********
**********
```

Se o segundo argumento for uma string, então o primeiro caractere dessa string é usado, isto é, se a linha 20 fosse substituída por:

```
20 A$=STRING$(B,"*")
```

o efeito seria o mesmo.

CAPÍTULO 21

MODOS DE TELA

Até agora você viu somente os modos de tela de texto 0 e 1. De fato existem mais dois modos de tela: o 2 e o 3. Iremos usá-los nos capítulos seguintes. Em primeiro lugar, um resumo das características de cada modo de tela.

OS MODOS DE TELA DE TEXTO (SCREEN 0 E 1)

Você pode alterar a largura de ambos os modos de tela usando o comando WIDTH (veja o Capítulo 9):

MODO DE TELA	LARGURA PADRÃO	LARGURA MÁXIMA	TAMANHO DE CARACTERE
SCREEN 0	37	40	6×8
SCREEN 1	29	32	8×8

Ambos os SCREENs contêm 24 linhas de texto. Apesar de você poder colocar mais caracteres dentro do SCREEN 0 que no SCREEN 1, o espaço alocado para cada caractere é menor. Apenas 6×8 pontos (ou pixels como também são chamados) são usados para cada caractere no SCREEN 0, enquanto são usados 8×8 pontos no SCREEN 1. Isto significa que os caracteres alfanuméricos aparecem mais comprimidos no

SCREEN 0 que no SCREEN 1, pois as últimas duas colunas de pontos nos quadrados de 8 × 8 pontos usadas no SCREEN 1, são deixadas em branco e são essas duas colunas que estão faltando nas posições de caractere do SCREEN 0. A compressão das posições de caracteres é mais óbvia quando se comparam caracteres gráficos.

Ao se ligar o computador, um desses SCREENs de texto é o SCREEN padrão.

O MSX é capaz de produzir 16 cores (veja o Capítulo 22 para mais detalhes). No SCREEN 1 você pode definir as cores de texto, fundo e das margens para qualquer combinação dessas 16 cores. Você não pode ter mais de uma cor de texto ao mesmo tempo. As cores padrões no Expert, são:

TEXTO:	BRANCA
FUNDO:	PRETO
MARGENS:	PRETO

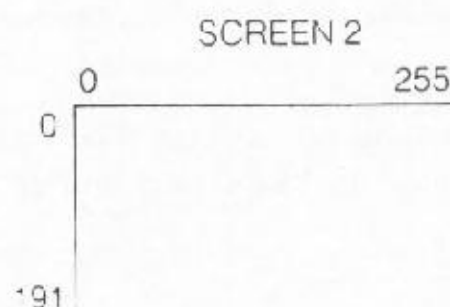
(No HOT-BIT, a cor do fundo e da moldura é azul-escuro e a cor das letras é branco.)

No SCREEN 0 você somente pode definir as cores de texto e de fundo, sendo as cores padrões as mesmas vistas acima.

Se em qualquer ponto você alterar a cor de texto, todo o texto do modo de tela muda de cor.

MODOS GRÁFICOS DE ALTA RESOLUÇÃO (SCREEN 2 E 3)

O SCREEN 2, é o modo gráfico de tela de alta resolução.

Ele tem 256 pontos de largura por 192 de comprimento. Cada ponto pode ser manipulado nesse SCREEN. O ponto do canto superior esquerdo está na posição de coordenada (0,0), e o ponto do canto inferior direito na posição (255,191):

Você não pode mudar a largura deste SCREEN. Se você emitir um comando WIDTH enquanto estiver no SCREEN 2, o computador irá lembrá-lo disso e, ao retornar para o modo de tela de texto, irá definir a largura como estava definida no SCREEN 2.

Você pode ter digitado:

```
SCREEN 2  <RETURN>
```

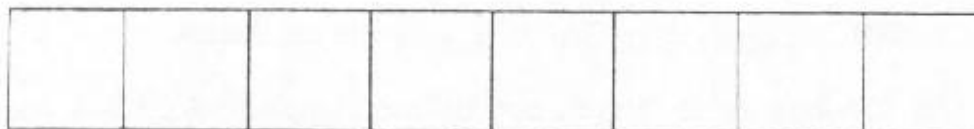
enquanto estava no modo direto e ser surpreendido por nada ter sido alterado. Isto ocorre devido ao fato de você não poder acessar modos gráficos de tela no modo direto. Você deve incluir o comando SCREEN em um programa para poder examinar modos gráficos de tela, isto é,

```
10  SCREEN 2
20  GOTO 20
```

A linha 20 congela o modo gráfico de tela; isto é necessário pois o computador retorna imediatamente para o modo de texto padrão ao completar a execução de um programa. Ela também irá retornar para o modo de tela de texto ao encontrar um comando INPUT em um programa, ou um erro.

Para parar este programa e retornar para o modo de tela de texto, pressione <CTRL><STOP>.

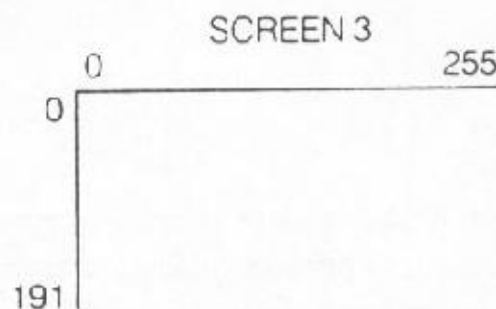
Os pontos no modo de tela são agrupados em grupos com 8 pontos consecutivos na horizontal. O primeiro grupo vai das coordenadas (0,0) a (7,0).

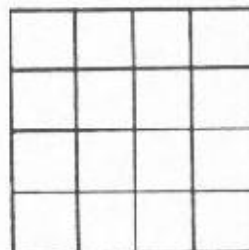
Um grupo 8 × 1 pontos:

Cada grupo de 8 pontos pode conter somente duas cores, uma cor de texto e uma cor de fundo. Linhas adjacentes podem conter duas cores diferentes. A resolução de cores deste modo de tela é 32 × 192.

Para imprimir caracteres nesse modo de tela, veja a seção avançada, Capítulo 14. Você não pode usar o comando PRINT diretamente, nos modos gráficos.

O MODO DE TELA DE MÚLTIPLAS CORES (SCREEN 3)

O SCREEN 3 é o SCREEN de múltiplas cores e pode ser usado para gráficos de baixa resolução.

Da mesma forma que o SCREEN 2, ele tem 256 pontos de largura por 192 de profundidade, mas você não pode manipular cada ponto desse modo de tela:

O menor ponto que você pode desenhar nesse modo de tela é um bloco de 4 × 4 pontos. Você pode ainda desenhar pontos na coordenada (0,0), mas irá descobrir que o bloco que aparece na tela também cobre os pontos (1,0), (2,0), (3,0) e (3,3). Dessa forma, ao se desenhar (3,3) o efeito seria o mesmo.

Um bloco de 4 × 4 pontos:

Cada bloco 4 × 4 pode conter somente uma cor, a cor do texto. Dessa forma, a resolução de cor desse SCREEN é de 64 × 48.

Da mesma forma que o SCREEN 2, você não pode acessar esse SCREEN no modo direto; você precisará congelar o SCREEN usando um loop infinito. Um comando INPUT ou um erro irá forçar você de volta para o modo de tela de texto padrão.

CAPÍTULO 22

CORES

Ao ligar o computador, são estas as cores que aparecem no modo de tela: um fundo preto com uma margem preta. O primeiro plano, ou texto, é branco. (No HOT-BIT a cor do fundo e da moldura é azul-escuro e a das letras é o branco.)

Você pode alterar qualquer uma dessas cores usando o comando COLOR.

Após ter recebido o comando COLOR, o computador espera por três números separados por vírgulas. O primeiro número define a cor do texto, o segundo, a cor do fundo e o último a cor das margens. Existem 16 cores para você usar:

0	Transparente
1	Preto
2	Verde-médio
3	Verde-claro
4	Azul-escuro
5	Azul-médio
6	Vermelho-escuro
7	Azul-claro (ou cian)
8	Vermelho-médio
9	Vermelho-claro
10	Amarelo-escuro
11	Amarelo-claro
12	Verde-escuro
13	Roxo
14	Cinza
15	Branco

de modo que a cor padrão é:

```
COLOR 15,1,1 no Expert e COLOR 15,4,4 no HOT-BIT
```

Para voltar às cores iniciais, basta pressionar F6.

A tecla F1 está definida para imprimir COLOR no modo de tela.

Você não precisa digitar todos os três números para o texto, fundo e margens quando estiver usando COLOR. Forneça somente um código de cor para aquela que você quiser alterar. Por exemplo:

```
COLOR 1
```

define a cor do texto como preto, deixando a cor do fundo e das margens como estão.

```
COLOR,15,15
```

A cor do fundo e das margens são definidas para branco, mantendo a cor do texto preta. Observe que, apesar da cor do texto não ter sido especificada, a vírgula que a segue deve ser escrita, pois ela diz ao computador que o número seguinte define o fundo.

```
COLOR ,,10
```

O comando acima define as margens para amarelo-escuro.

Você precisa estar no modo de tela (SCREEN) 1, 2 ou 3 para notar isso, pois o modo de tela 0 não tem margens.

Se você alterou a cor de fundo enquanto estava em qualquer um dos modos gráficos de tela, precisará executar o comando CLS para notar o efeito. Ele apaga a tela e coloca nova cor de fundo.

O programa seguinte exibe as combinações de cores das margens e de fundo disponíveis no MSX. Digite-o no modo de tela 1:

```
10    REM cores
20    FOR MG=0 TO 15
30    COLOR,,MG
40    FOR FU=0 TO 15
```

```
50      COLOR,FU
60      FOR LOOP=1 TO 200 :NEXT LOOP
70      NEXT FU
80      NEXT MG
90      COLOR 15,4,7
```

O loop na linha 60 dá a você tempo para ver as diferentes combinações de cores; sem essa linha, as cores iriam piscar muito rapidamente para serem vistas de forma apropriada.

Observe que você pode usar variáveis numéricas e expressões numéricas no comando COLOR. Se você usar um número real no comando COLOR, ele será truncado para o inteiro mais próximo. O número deve estar num limite entre 0 e 15.

Tente executar esse programa nos outros modos de tela. Você precisará inserir um comando CLS para modos de tela 2 e 3.

Já pensou no que a cor 0 faz?

Ela define a cor de fundo, ou do texto, como sendo a mesma cor das margens. Se usada para definir cor de margens, estas tornam-se pretas.

EXERCÍCIO

1. Defina uma das teclas de função para dar uma combinação diferente de cores para o texto, fundo e margens.

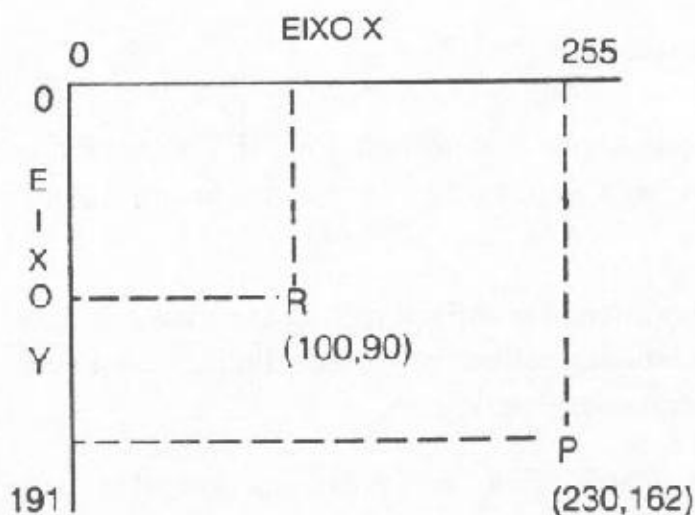
CAPÍTULO 23

DESENHANDO PONTOS

Os dois primeiros comandos tratados neste capítulo, PSET e PRESET, são muito similares. Por serem comandos gráficos, eles podem ser somente usados nos modos de tela 2 e 3.

PRESET pode ser usado para fixar um ponto na cor de fundo. As coordenadas do ponto podem ser absolutas ou relativas.

Após usar PRESET, você descobrirá que a cor do texto é a mesma que a do fundo.

Tanto no modo de tela (SCREEN) 2 como no modo de tela 3, a coordenada x deve estar no limite de 0 a 255, e a coordenada y entre 0 e 191. As coordenadas absolutas são lidas diretamente nos eixos X e Y:

O ponto R está na coordenada x = 100, e na coordenada y = 90. P está na coordenada absoluta (230,162).

Uma coordenada relativa fornece a posição de um ponto em relação a outro ponto de referência. Por exemplo, o ponto P está na coordenada relativa (130,72) com relação a R. Isto significa que ele está 130 pontos mais à direita no eixo X do que R e 72 pontos mais abaixo no eixo Y.

Em todos os comandos gráficos, o ponto de referência é sempre tomado em relação à última posição do cursor gráfico.

A posição do cursor gráfico pode ser definida com PRESET:

```
10  SCREEN 2
20  PRESET (100,100)
30  GOTO 30
```

Quando você executar o programa, pensará que nada ocorreu! Entretanto, agora o cursor gráfico está posicionado no ponto (100,100). Qualquer comando gráfico posterior que use coordenadas relativas, irá usar o ponto (100,100) como ponto de referência.

Se o comando gráfico seguinte desenhar uma linha, por exemplo, você deve especificar a cor da linha, ou redefinir a cor dos caracteres em primeiro lugar, usando o comando COLOR. De outra forma, a sua linha ficaria invisível!

Se você especificar uma dada cor em um comando PRESET, um ponto dessa cor será desenhado e essa cor passará a ser a cor dos caracteres. Substitua a linha 20 por:

```
20  PRESET (100,100),1
```

O número que segue o comando PRESET deve ser um dos códigos de cor, conforme especificado no Capítulo 22. Se for usado um número real, a parte decimal será ignorada.

Se as suas coordenadas estiverem fora do modo de tela, obviamente você não verá nada! Se as coordenadas estiverem fora do limite de números inteiros de –32768 a 32767, você obterá uma mensagem de erro.

O comando PSET fixa um ponto na cor dos caracteres na coordenada específica. Essas coordenadas podem ser absolutas ou relativas.

Se uma cor for especificada no comando, então o ponto será desenhado nesta cor. Da mesma forma que com o PRESET, este comando também altera a cor de fundo para a cor especificada.

O programa seguinte desenha blocos de cores aleatoriamente no modo de tela 3 usando o comando PSET. Esses blocos são, então, apagados ao desenhar os mesmos pontos aleatórios na cor de fundo usando o PRESET.

```
10 REM pontos
20 R=TIME
30 RDRST=RND(-R)
40 SCREEN 3
50 REM desenha pontos coloridos
60 FOR I%=1 TO 300
70 GOSUB 1000
80 PSET(X%,Y%),Z%
90 NEXT I%
100 REM coloca pontos na cor do fundo
110 RDRST=RND(-R)
120 FOR J%=1 TO 300
130 GOSUB 1000
140 PRESET(X%,Y%)
150 NEXT J%
160 END
990 REM sub-rotina randomica
1000 X%=RND(1)*256
1010 Y%=RND(1)*192
1020 Z%=RND(1)*16
1030 RETURN
```

Variáveis inteiras são usadas ao longo deste programa para aumentar a velocidade de execução (isto é, usamos X% em vez de X).

Cada vez que o programa for executado, os pontos serão desenhados em posições aleatórias diferentes. Isto ocorre devido ao fato de que os pontos aleatórios são definidos pela variável TIME e essa variável assume um valor diferente cada vez que o programa é executado. Os mesmos pontos aleatórios são gerados na segunda metade do programa. Para gerar os mesmos números, usamos o mesmo argumento negativo da função RND na linha 110 e na linha 30.

Finalmente, o comando POINT retorna o código da cor de um ponto em qualquer um dos modos gráficos de tela. O número retornado está, dessa forma, dentro do

limite de 0 a 15. Será retornado o valor –1 se as coordenadas do comando POINT estiverem fora do modo de tela. Digite:

```
10 COLOR,2
20 SCREEN 2
30 P=POINT(100,100)
```

Agora, de volta ao modo direto digite:

```
PRINT P
```

O número 2 será exibido, pois esse é o código de cor para o verde-médio, que é a cor de fundo atual.

EXERCÍCIO

1. Tente desenhar a curva da função seno, usando:

```
Y=A*SIN(X)+A
```

Observe que A define a amplitude da curva. Varie X na faixa de 0 a 2*PI.

CAPÍTULO 24

DESENHANDO LINHAS E RETÂNGULOS

Este capítulo trata do comando LINE, um outro comando gráfico que pode ser usado somente nos SCREENs 2 e 3.

DESENHANDO LINHAS

Para desenhar uma linha, você deve especificar a posição inicial e final; isso é feito usando ou coordenadas absolutas ou relativas (ou mesmo uma mistura das duas). Digite este programa:

```
10 SCREEN 2
20 LINE(0,0)-(255,191)
30 GOTO 30
```

Este programa desenha uma linha diagonal ao longo do modo de tela. Observe o sinal "–" entre os dois pares de coordenadas.

Foram usadas coordenadas absolutas. O primeiro par de coordenadas, (0,0), fornece o ponto inicial da linha; (255,191) fornece a posição final da linha.

Altere a linha 10 deste programa de modo que ele desenhe no modo de tela 3, para comparar a resolução dos dois modos gráficos de tela.

Se você tentar usar coordenadas x ou y maiores que 255 ou 191 respectivamente, o computador desenhará até o canto do modo de tela, atribuindo a x o valor 255 e a y o valor 191. Entretanto, se as coordenadas estiverem fora do limite permitido, –32768 a 32767, você receberá uma mensagem de erro.

Você não precisa especificar o ponto inicial da linha. Se este não for especificado, o computador começará a desenhar a partir da posição atual do cursor gráfico.

Quando você acaba de desenhar uma linha, o cursor fica na posição final da mesma.

Você pode usar coordenadas relativas para desenhar a mesma linha. Substitua a linha 20 do programa por:

```
20  LINE (0,0)–STEP (255,191)
```

O comando STEP diz ao computador que as coordenadas a seguir são relativas. Neste caso, a posição final de x será 255 pontos para a direita no eixo X a contar do ponto inicial, e a posição final y estará 191 pontos abaixo da posição inicial no eixo Y, colocando o último ponto da linha na coordenada absoluta (255,191).

LINHAS COLORIDAS

Até o momento você desenhou somente linhas na cor atual do texto. Entretanto, é possível especificar a cor da linha, seguindo o comando LINE com uma vírgula e então por um código de cor válido, isto é, um número entre 0 e 15. Vá ao Capítulo 22 para ver os códigos de cores. Agora substitua a linha 20 do último programa por:

```
20  LINE–(255,191),10
```

Agora o programa desenha uma linha diagonal amarela ao longo do modo de tela.

O próximo programa demonstra a resolução de cores no modo de tela 2:

```
10  SCREEN 2
20  FOR A=0 TO 15
30  LINE (0+A,0)-(200+A,191),A
40  NEXT A
50  GOTO 50
```

Você verá 16 linhas diagonais no modo de tela com as cores "borradas".

Substitua a linha 30 por:

```
30  LINE(0,0+A)-(255,0+A),A
```

e conseguirá ver as cores sem borrões.

Obtivemos esses efeitos devido ao modo como o modo gráfico de tela é armazenado na memória do micro.

Cada linha horizontal é dividida em grupos com 8 pontos de comprimento. Cada grupo pode conter apenas duas cores ao mesmo tempo; uma de texto e outra de fundo. Ao se desenhar a primeira linha na diagonal, a cor de texto é definida como 0 e a linha é desenhada nessa cor. A linha seguinte muda a cor de texto para 1, mas isso altera a cor de texto para todo o grupo de 8 pontos, de modo que o primeiro ponto da primeira linha muda de cor para preto. A oitava linha desenhada altera a cor de texto do primeiro grupo de 8 pontos pela última vez; este grupo completo tem agora a cor azul-cian.

A resolução vertical de cores é melhor que a resolução horizontal. A cor de cada ponto vertical é independente da cor do ponto de cima ou de baixo e, por isso, as 16 linhas horizontais não interferem umas com as cores das outras.

Tente executar os dois programas no modo de tela 3.

Nesse modo de tela, você verá as mesmas quatro linhas a cada vez, já que a resolução das cores é igual. Você pode ter uma cor para cada bloco de 4×4 pontos. A resolução de cor é, dessa forma, a mesma que para a resolução gráfica. As primeiras quatro linhas são todas desenhadas umas sobre as outras, de modo que você somente verá a última que é verde-claro.

DESENHANDO QUADRADOS

É possível desenhar um quadrado usando quatro comandos LINE um após o outro. Entretanto, existe uma maneira mais fácil de desenhar quadrados se você conhecer as coordenadas dos cantos superior esquerdo e inferior direito. Você continuará usando o comando LINE, mas agora o primeiro par de coordenadas fornece a localização do canto superior esquerdo, e o segundo, a localização do canto inferior direito. Para informar ao computador que você quer um quadrado e não uma linha, você deve escrever a letra "B" após o código da cor. Digite o seguinte programa:

```
10 REM caixas
20 SCREEN 2
30 FOR A=0 TO 90 STEP 2
40 LINE(5+A,5+A)-(255-A,185-A),A/10,B
50 NEXT A
60 GOTO 60
```

Este programa irá desenhar dez quadrados de tamanhos e cores diferentes, incluindo a cor transparente.

Observe que você pode usar uma expressão no lugar do código de cor, mas essa expressão deve ser um número entre 0 e 15. Se ela for um número real, a parte decimal será anulada.

Veja o efeito de executar este programa no modo de tela 3, alterando a linha 20 para:

```
20 SCREEN 3
```

Se você conhecer os comprimentos dos lados dos quadrados, da mesma forma que sabe as coordenadas de seus cantos, é aconselhável o uso de coordenadas relativas para o segundo par de coordenadas, ou seja, para desenhar um retângulo de 100 pontos de comprimento na direção X e 50 pontos de largura na direção Y, você pode usar:

```
10 SCREEN 2
20 LINE(10,10)-STEP(100,50),9,B
30 GOTO 30
```

O programa desenha um retângulo nas dimensões corretas. O canto superior esquerdo está situado na coordenada absoluta (10,10).

O segundo par de coordenadas está agora especificando os tamanhos dos lados do retângulo que são paralelos aos eixos X e Y, respectivamente.

Este é o melhor método para desenhar quadrados. Por exemplo:

```
10 REM quadrados
20 SCREEN 2
30 FOR A=10 TO 90
40 LINE (A,A)-STEP(A,A),,B
50 NEXT A
60 GOTO 60
```

O último programa desenha quadrados coloridos de tamanhos crescentes na diagonal do modo de tela.

QUADRILÁTEROS COLORIDOS

Para desenhar um quadrilátero e pintá-lo com uma cor específica, substitua o "B" por "BF". A cor especificada no comando LINE será a usada para pintar o quadrilátero.

```
10 REM CAIXA VERMELHA
20 SCREEN 2
30 LINE (20,20)-STEP (100,100),9,BF
40 GOTO 40
```

EXERCÍCIO

1. Escreva um programa que desenhe um quadrado de um tamanho aleatório no modo de tela. Pinte esse quadrado com a cor verde.

CAPÍTULO 25

DESENHANDO CÍRCULOS E ELIPSES

O comando CIRCLE permite a você desenhar círculos e elipses, tanto inteiros como parcialmente. Você deve especificar a posição do centro do círculo, em coordenadas absolutas ou relativas, e o comprimento do raio. Como opção, você pode especificar a cor do círculo e a parte do círculo que deve ser desenhada. CIRCLE é um comando gráfico e pode ser usado somente nos modos gráficos de tela.

DESENHANDO CÍRCULOS

Para desenhar um círculo, você deve especificar a coordenada central do círculo e o tamanho do seu raio. Por exemplo:

```
10 REM desenhando círculos
20 SCREEN 2
30 CIRCLE(128,100),90
40 GOTO 40
```

O programa acima desenha um círculo centrado na coordenada (128,100) com um raio de 90 pontos, na cor de texto atual. Obviamente, se você especificar um raio muito maior que o modo de tela, não poderá ver o seu círculo, ou verá somente a parte dele que couber no modo de tela.

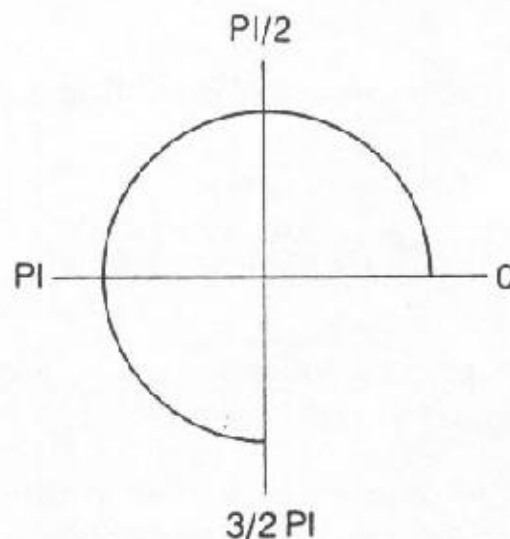
Para especificar a cor do círculo, coloque uma vírgula após o raio seguida de um código de cor válido, ou seja, um número entre 0 e 15 (veja o Capítulo 22 para detalhes sobre códigos de cores). O programa a seguir desenha um círculo em vermelho:

```
10 SCREEN 2
20 CIRCLE (196,199),90,9
30 GOTO 30
```

DESENHANDO ARCOS

Para desenhar apenas parte de um círculo, ou um arco, você deve especificar o ângulo inicial e final. O programa seguinte desenha três quartos de um círculo na cor preta:

```
10 REM desenhando arcos
20 SCREEN 2
30 PI=4*ATN(1)
40 CIRCLE(128,100),90,9,0,3*PI/2
50 GOTO 50
```

Os dois últimos números que incluímos fornecem os ângulos inicial e final do arco. Os ângulos devem ser especificados em radianos. (Se você não se lembrar de o que são radianos, veja o Capítulo 19.) A linha 30 serve apenas para calcular o valor de PI, de modo que o arco seja desenhado entre os ângulos 0 e 3/2*PI radianos:

Todos os ângulos são medidos a partir do eixo X no sentido anti-horário. Como o círculo tem 2*PI radianos, os ângulos especificados podem ter qualquer valor entre 0 e 2*PI.

DESENHANDO PIZZAS

Para fazer isso, coloque um sinal "–" antes do ângulo inicial e final. Isso resultará em linhas sendo desenhadas a partir do centro do círculo e terminando no arco. Execute o seguinte programa:

```
10 REM DESENHANDO PIZZAS
20 SCREEN 2
30 PI = 4 * ATN(1)
40 CIRCLE (128,100),90,,-PI/2,-PI
50 GOTO 50
```

Você deve ver o setor superior esquerdo do círculo desenhado na cor atual do texto.

DESENHANDO ELIPSES

Para desenhar uma elipse, você deve especificar a relação entre os raios. Isso é definido por:

$$\text{RELAÇÃO DE RAIOS} = \frac{\text{RAIO VERTICAL}}{\text{RAIO HORIZONTAL}}$$

Se a relação entre os raios for maior que 1, então a elipse se alongará na vertical. O raio especificado será o raio vertical.

Se a relação entre os raios for menor que 1, então a elipse será alongada na horizontal e o raio especificado será o raio horizontal.

O programa a seguir desenha uma elipse completa, pois os ângulos inicial e final não foram especificados:

```
10 REM desenhando elipses
20 SCREEN 2
30 CIRCLE(128,100),90,9,,,1.5
40 GOTO 40
```

Como a relação entre os raios é 1.5, a elipse será alongada no sentido vertical e o raio vertical será de 90 pontos.

Agora substitua a linha 30 por:

```
30 CIRCLE(128,100),90,1,,,0.5
```

Você verá uma elipse alongada na horizontal, com raio horizontal de 90 pontos.

Tente relações entre os raios 100 e 0.01. Com essas relações de raio, as elipses terão o aspecto de linhas verticais e de linhas horizontais, respectivamente, com raios iguais a 180 pontos.

O programa seguinte desenha elipses coloridas, todas com relações de raio aleatórias, sempre posicionadas no centro do modo de tela:

```
10 REM elipses diversas
20 SCREEN 2
30 FOR B=1 TO 15
40 A=RND(-TIME)*2
50 CIRCLE(128,96),90,B,,,A
60 NEXT B
70 GOTO 70
```

Se você desejar, pode desenhar parte de uma elipse, especificando os ângulos inicial e final, da mesma forma como fez ao desenhar círculos. O sinal "–", de prefixo, também deve ser usado:

```
10 SCREEN 3
20 PI=4*ATN(1)
30 FOR B=1 TO 15
40 A=B/10
```

```
50 SA=RND(-TIME)*2*PI
60 EA=RND(-TIME)*2*PI
70 CIRCLE(126,96),90,B,SA,EA,A
80 NEXT B
90 GOTO 90
```

Este programa desenha figuras helicoidais centradas em (126,96). Os ângulos inicial e final são calculados usando a função RND. Observe que o ângulo final pode ser menor que o ângulo inicial. A relação de raio oscila na faixa de 0.1 a 1.5. Execute o programa no modo de tela 2 para comparar a resolução dos modos gráficos de tela (não se esqueça de alterar a linha 10 para: 10 SCREEN 2).

EXERCÍCIOS

1. Escrèva um programa que desenhe 10 pequenos círculos em posições aleatórias do modo de tela.
2. Altere o seu programa de modo que os círculos tenham raios aleatórios.
3. Desenhe um círculo composto de quatro setores, cada qual sendo desenhado em uma cor diferente.

CAPÍTULO 26

A MACROLINGUAGEM GRÁFICA

A macrolinguagem gráfica simula o movimento de uma caneta no papel e permite a você desenhar figuras complicadas com facilidade, usando o comando DRAW. Você poderá mover o cursor gráfico para qualquer ponto do modo de tela, com a "caneta" em duas posições: "para cima" ou "para baixo".

O comando DRAW deve ser seguido por uma string. Essa string contém macrocomandos gráficos que são representados por caracteres simples.

Para desenhar uma linha em uma das quatro direções, isto é, para cima, para baixo, para a direita ou para a esquerda, devem ser usados os seguintes macrocomandos gráficos:

U	Desenha para cima.
D	Desenha para baixo.
L	Desenha para a esquerda.
R	Desenha para a direita.

Você deve especificar o tamanho da linha logo após o comando. O comprimento é medido em pontos.

O programa a seguir demonstra como usar a macrolinguagem:

```
10 REM QUADRADO
20 SCREEN2
30 PSET(123,91)
40 DRAW"r10d10l10u10"
50 GOTO 50
```

Ao ver a linha 40, o computador executa os macrocomandos seqüencialmente ou seja, ele desenha uma linha para a direita com o comprimento de 10 pontos, então vai para baixo 10 pontos, para a esquerda mais 10 pontos e, finalmente, para cima por mais 10 pontos.

Podem ser traçadas linhas diagonais usando os quatro comandos seguintes:

E Desenha diagonalmente para cima e para a direita (↗).
F Desenha diagonalmente para baixo e para a direita (↘).
G Desenha diagonalmente para baixo e para a esquerda (↙).
H Desenha diagonalmente para cima e para a esquerda (↖).

Outra vez, cada um desses comandos deve ser seguido por um número. Entretanto, neste caso o número especifica o número de pontos movidos nas direções x e y.

Por exemplo, a coordenada relativa do último ponto da linha desenhada por E10 poderia ser (10,10), tomando o primeiro ponto da linha como referência.

Este programa desenha um hexágono:

```
10 REM HEXAGONO
20 SCREEN2
30 PSET(128,96)
40 DRAW"e20f20d30g20h20u30
50 GOTO 50
```

O comando M é usado para desenhar uma linha em uma posição de coordenada específica no modo de tela. Esse comando deve ser seguido pelos valores de x e y da posição da coordenada do último ponto da linha. Os valores x e y podem ser relativos ou absolutos. Por exemplo:

M40,50

irá mover o cursor gráfico para a posição de coordenada absoluta (40,50).

Para especificar coordenadas relativas, são usados os prefixos + ou – :

M+40,+50 Desenha por uma posição afastada +40 pontos no eixo X e +50 pontos do eixo Y.
M+40,–50 Desenha por uma posição afastada +40 pontos no eixo X, e –50 pontos no eixo Y.

Observe que os comandos a seguir são equivalentes:

M + 0, −10 = U10	M + 10, −10 = E10
M + 0, 10 = D10	M + 10,10 = F10
M + 10,0 = R10	M − 10,10 = G10
M − 10,0 = L10	M − 10, −10 = H10

No último exemplo de programa, o comando PSET foi usado para posicionar o cursor gráfico antes de iniciar o desenho. De fato, também se pode fazer isso usando o prefixo B antes do comando M. B significa branco. Esse prefixo faz o cursor se mover sem desenhar:

```
30  DRAW"BM128,96"
```

pode ser usado dentro de um programa para posicionar o cursor no centro do modo de tela.

O prefixo B também pode ser usado com outros macrocomandos que você já viu, o que era esperado já que esses comandos são variações do comando M.

Se você desejar mover o cursor de volta à posição inicial após desenhar uma linha, use o prefixo N.

O programa seguinte desenha uma estrela no modo de tela:

```
10 REM ESTRELA
20 SCREEN2
30 REM MOVE CURSOR
40 DRAW"BM128,96"
50 REM DESENHA BRACOS
60 DRAW"NU50ND50NR50NL50"
70 DRAW"NE30NF30NG30NH30"
80 GOTO 80
```

A cada vez que um braço da estrela for desenhado, o cursor se move de volta para o centro da estrela.

Agora você é capaz de desenhar figuras bastante complicadas. Uma vez definida a sua figura, é possível alterar a sua orientação no modo de tela usando o comando A (Ângulo).

Esse comando roda os eixos do modo de tela, no sentido anti-horário, para 0, 90, 180 ou 270 graus, a partir da orientação padrão. Por sua vez, isso afeta a orientação a direção dos comandos U, D, L, R, F, E, G e H.

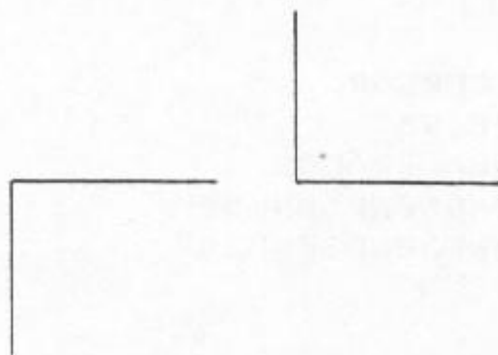
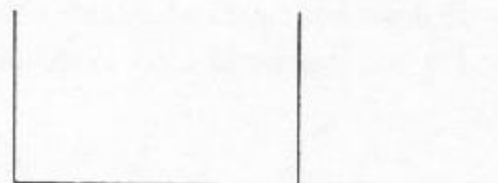
O comando A deve ser seguido por um número inteiro entre 0 e 3, onde:

An Rotação.
A0 Reposiciona os eixos na posição padrão.
A1 Executa a orientação padrão dos eixos no sentido anti-horário em 90 graus.
A2 Executa a orientação padrão dos eixos no sentido anti-horário em 180 graus.
A3 Executa a orientação padrão dos eixos no sentido anti-horário em 270 graus.

Uma vez executado o comando A, a orientação de todos os comandos de direção seguintes no programa será relacionada ao novo eixo. Para retornar ao valor padrão, você deve usar A0 com o comando DRAW.

Execute o seguinte programa duas vezes:

```
10 REM ANGULO
20 SCREEN2
30 DRAW"BM50,100NR50ND50
40 DRAW"A1BM150,100NR50ND50"
50 GOTO 50
```

Na primeira vez você verá:

Na segunda vez:

Isso ocorre porque na primeira vez o comando A ainda está definido para A0 na primeira execução. Na segunda execução A está definido como A1.

Se você quiser repetir o primeiro resultado, redefina o ângulo de rotação para A0 no início da string da linha 30.

Até o momento você somente desenhou na cor de texto. Para desenhar em uma cor diferente, ou seja, para alterar a cor de fundo, você deve usar o comando C. Ele deve ser seguido por um código de cor. Os códigos de cores são os mesmos que os usados com o comando COLOR, já visto no Capítulo 22. Desse modo:

C0	Transparente	C8	Vermelho-médio
C1	Preto	C9	Vermelho-claro
C2	Verde-médio	C10	Amarelo-escuro
C3	Verde-claro	C11	Amarelo-claro
C4	Azul-escuro	C12	Verde-escuro
C5	Azul-médio	C13	Roxo
C6	Vermelho-escuro	C14	Cinza
C7	Azul-claro	C15	Branco

Da mesma forma que com o comando A, uma vez definido o C, a cor atual de texto permanecerá até que seja redefinida ou por um novo comando C ou um COLOR.

O comando S altera a escala, ou tamanho do seu desenho. O número n, que segue a S, define o fator de escala. Esse número pode ser qualquer inteiro entre 0 e 255.

O fator de escala (SF) é definido como:

SF = n/4

Dessa forma, S1 resultará em um SF de 1/4. Todos os tamanhos especificados pelos comandos gráficos seguintes serão colocados na escala de 1/4. Execute o programa seguinte:

```
10 REM escala
20 SCREEN2
30 DRAW"bm50,50"
40 DRAW"r150d100l150u100"
50 DRAW"bm50,50"
60 DRAW"s1r150d100l150u100"
70 GOTO 70
```

Você verá dois retângulos, um deles com 1/4 do tamanho do outro. Se você executar o programa outra vez, os retângulos serão desenhados uns sobre os outros, em tamanhos reduzidos.

Para redefinir SF para 1, você pode usar S4 ou S0. Insira um dos dois no início da string da linha 40 e execute o programa outra vez.

Se você desejar desenhar a mesma figura mais de uma vez em um programa, é aconselhável colocar a string que contenha os macrocomandos necessários em uma variável string. O programa a seguir desenha quatro "diamantes" no modo de tela:

```
10 REM diamantes
20 SCREEN2
30 A$="f40g40h40e40"
40 DRAW"bm50,8"
50 DRAWA$
60 DRAW"bm200,8"
70 DRAWA$
80 DRAW"bm50,104"
90 DRAWA$
100 DRAW"bm200,104"
110 DRAWA$
120 GOTO 120
```

O programa pode ser encurtado consideravelmente, colocando mais de um comando em cada comando DRAW. Para colocar uma string dentro de um DRAW, deve-se usar o comando X. O programa abaixo desenha também quatro diamantes mas, desta vez, eles têm tamanhos diferentes. A string A$ é usada como substring:

```
10 REM diamantes em escala
20 SCREEN2
30 A$="f40g40h40e40"
40 DRAW"s4bm50,8xa$;"
50 DRAW"s3bm200,8xa$;"
60 DRAW"s2bm50,104xa$;"
70 DRAW"bm200,104xa$;"
100 GOTO 100
```

Observe que o nome da string é seguido por um ponto e vírgula; ele não pode ser omitido.

Se você desejar usar uma variável numérica dentro de uma string DRAW, talvez para definir o comprimento da linha, deve-se preceder o nome da variável pelo sinal "=". Outra vez, o nome deve ser seguido por ponto e vírgula. O programa abaixo desenha 15 quadrados e usa a variável i para definir a escala e a cor de cada quadrado.

```
10 REM INCREMENTANDO QUADRADOS
20 SCREEN 2
30 DRAW "BM31,186"
40 FOR X=1 TO 15
50 DRAW "S=X;C=X;U48R48D48L48"
60 NEXT X
70 GOTO 70
```

Os cantos verticais esquerdos dos quadrados têm de ser posicionados cuidadosamente para permanecer na última posição do ponto de um grupo horizontal de 8 pontos. Isto é feito para evitar manchas entre a última linha branca vertical e as 14 linhas horizontais que definem a parte superior dos outros quadrados. O efeito de mancha irá aparecer se você tentar usar mais de duas cores em um grupo horizontal de 8 pontos (veja o Capítulo 21 para mais detalhes). Para ver o efeito de mancha substitua a linha 30 por:

```
30 DRAW"BM25,186"
```

EXERCÍCIOS

1. Use os comandos S e C para desenhar estrelas aleatoriamente no modo de tela. Cada estrela deve ter cor e tamanho diferentes.

2. Desenhe uma casa.

CAPÍTULO 27

PINTANDO

O comando PAINT pode ser usado para preencher uma área que esteja completamente delimitada por uma linha de uma cor específica. As coordenadas do comando PAINT devem ser as coordenadas de um ponto dentro da região da figura a ser pintada. Elas podem ser relativas ou absolutas.

As restrições quanto às cores permitidas são um pouco diferentes nos dois modos gráficos de tela.

MODO DE TELA 2

Antes de usar o PAINT, você deve desenhar o esboço de uma figura usando um dos comandos gráficos LINE, CIRCLE ou DRAW. Neste caso, a cor usada no PAINT deve ser a mesma que a especificada nos comandos gráficos. Por exemplo:

```
10 REM paralelogramo pintado
20 SCREEN2
30 PRESET(100,100)
40 LINE-STEP(-50,50),6
50 LINE-STEP(100,0),6
60 LINE-STEP(50,-50),6
70 LINE-STEP(-100,0),6
80 PAINT(110,110),6
90 GOTO 90
```

As coordenadas (110,110) estão dentro do paralelogramo. O esboço de um paralelogramo vermelho será desenhado e, então, preenchido com tinta vermelha. Se você tentar pintar o paralelogramo com qualquer outra cor, a pintura irá transbordar e preencher todo o modo de tela.

Se o esboço tivesse sido desenhado na cor de texto atual, não seria necessário especificar a cor no comando PAINT, pois quando não for especificada a cor nesse comando, a cor atual de texto será usada. Por exemplo:

```
10 REM CIRCULO PINTADO
20 SCREEN 2
40 COLOR 10
50 CIRCLE (100,100),50
60 PAINT STEP(0,0)
70 GOTO 70
```

A linha 40 define a cor de texto como amarelo. O esboço do círculo é, dessa forma, desenhado em amarelo e pintado nessa mesma cor.

Observe que as coordenadas do comando PAINT são aquelas do centro do círculo.

No modo de tela 2, deve-se tomar cuidado para evitar efeitos de manchas quando do uso do comando PAINT. O programa abaixo desenha dois triângulos, um de costas para o outro. Um dos triângulos é então pintado de amarelo e o outro de vermelho:

```
10 REM triangulos v1
20 SCREEN2
30 COLOR10
40 PSET(48,50)
50 DRAW"a0s1d96r96h96"
60 PAINT(60,70)
70 COLOR9
80 PSET(70,60)
90 DRAW"s3r96d96h96"
100 PAINT(120,100)
110 GOTO 110
```

Quando você executar este programa notará um efeito de zigue-zague. Isto ocorre porque no modo de tela 2 cada grupo horizontal de 8 pontos não pode ter mais que duas cores. Não haverá problema ao se pintar o triângulo em amarelo, pois o texto é amarelo e o fundo é preto. Entretanto, quando o esboço do triângulo vermelho estiver

sendo desenhado, a cor do fundo ainda será o preto, e a nova cor de texto será o vermelho. A maioria dos triângulos amarelos não é afetada, mas onde um grupo de 8 pontos permanece em ambos os triângulos, a cor do grupo completo é alterada para vermelho. Isto resulta em pequenas seções do triângulo amarelo sendo perdidas para o triângulo preto, e daí o efeito de zigue-zague.

Existe um remédio simples para este problema. Execute o seguinte programa:

```
10 REM triangulos v2
20 SCREEN 2
30 COLOR 10
40 PSET(48,50)
50 DRAW"d96r96u96l96"
60 PAINT(60,70)
70 COLOR 9
80 PSET(48,50)
90 DRAW"r96d96h96"
100 PAINT(70,60)
110 GOTO 110
```

Neste programa, um quadrado amarelo é desenhado e pintado. Um triângulo vermelho é então desenhado e pintado no topo dele. Como o quadrado contém apenas uma cor, não existem problemas quando se desenha o triângulo vermelho.

MODO DE TELA 3

Neste modo de tela, você pode pintar uma figura em qualquer cor, independente da cor usada na margem da figura.

```
10 REM pintando com uma margem
20 SCREEN3
30 LINE(20,20)-(100,100),7,B
40 PAINT(50,50),9,7
50 GOTO 50
```

São dados dois números após o comando PAINT. O primeiro número fornece a cor do PAINT – que pode ser qualquer cor. O segundo número fornece a cor da moldura. A cor da moldura é a cor da figura. No exemplo acima, o quadro foi originalmente desenhado em azul-cian, de modo que a cor da moldura também é azul-cian. Se você tentar usar qualquer outra cor para a margem, a pintura irá transbordar e preencher todo o modo de tela.

Se o esboço da figura foi desenhado na cor de texto, então você deve definir a cor de pintura do PAINT igual a essa cor de texto; isto deve ser feito sempre que você especificar uma cor diferente da margem para pintar o restante da figura. Execute o programa a seguir:

```
10 REM triangulo
20 SCREEN3
30 COLOR15
40 LINE(20,20)-STEP(200,100)
50 LINE-STEP(-200,0)
60 LINE-(20,20)
70 PAINT(50,80),10,15
80 GOTO 80
```

Você verá um triângulo desenhado em branco e então pintado de amarelo com uma margem branca. Se a margem pintada não tivesse sido definida para branca no programa acima, a pintura teria coberto totalmente o modo de tela.

Entretanto, se você quiser um triângulo completamente branco, substitua a linha 70 por:

```
70 PAINT(50,80)
```

EXERCÍCIOS

1. Escreva um programa que desenhe um quadro preto no modo de tela, usando a opção "BF" nos comandos LINE e PAINT. Execute esse programa tanto no modo de tela 2 como no 3.

2. Escreva um programa que desenhe círculos concêntricos multicoloridos, cada qual sendo preenchido com uma cor diferente. Não se esqueça: para evitar os efeitos de mancha, inicie com maior círculo.

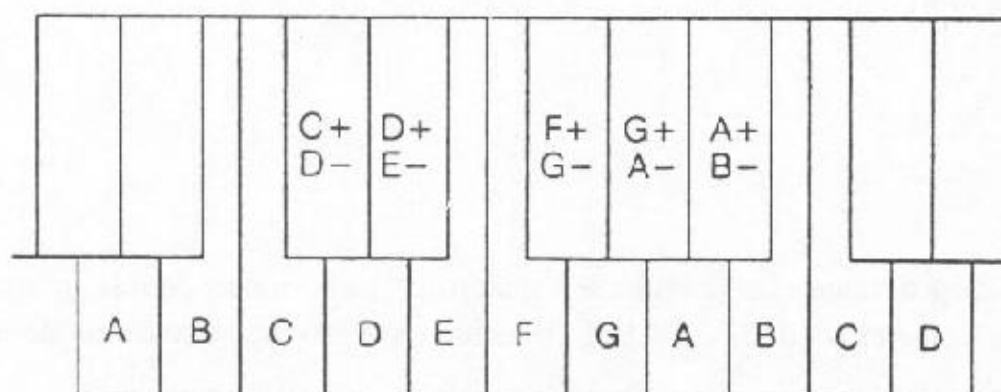
CAPÍTULO 28

A MACROLINGUAGEM MUSICAL

Se você tiver familiaridade com a notação musical, o comando PLAY irá permitir escrever ou transcrever músicas de uma pauta muito facilmente.

A macrolinguagem musical é muito similar à macrolinguagem gráfica vista no Capítulo 26.

O comando PLAY é seguido por uma string contendo os vários comandos permitidos na macrolinguagem musical.

Os comandos A, B, C, D, E, F e G tocam as notas musicais correspondentes. Qualquer um desses comandos pode ser seguido por um sinal "+" ou "–" para indicar se a nota correspondente é sustenido ou bemol. Entretanto, a nota sustenido ou bemol resultante deve corresponder à tecla preta do piano; B+, dessa forma seria inválido.

Agora vamos tocar as notas uma oitava acima. Para selecionar a oitava desejada, usamos o comando O. A primeira nota na oitava é C e a última é B. O comando O(oitava) deve ser seguido por um número entre 1 e 8. A oitava padrão é O4, na qual a primeira nota, C, corresponde ao "C" (dó) do meio no piano. Usando O1 a O8 você tem acesso a qualquer nota do piano.

O seguinte programa toca a escala do MI maior, duas oitavas acima:

```
10 REM Mi Maior:Notacao A-G
20 PLAY"ef+g+abo5c+d+ef+g+abo6c+d+e"
```

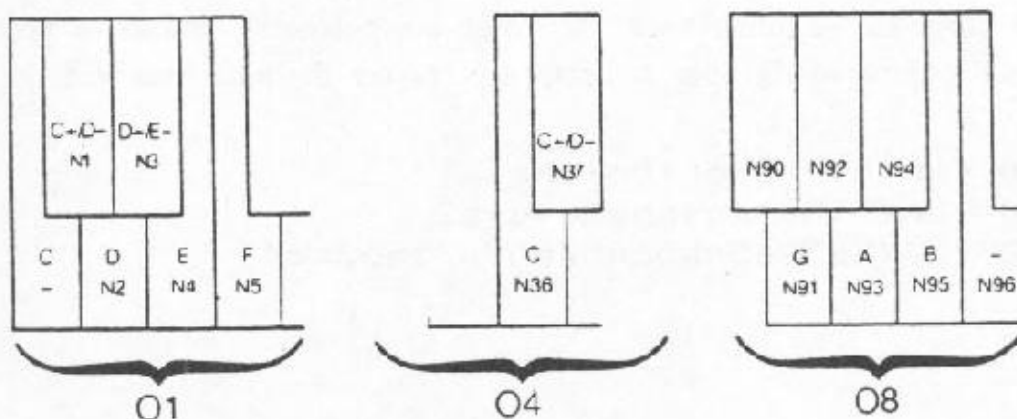
Se você executar esse programa novamente, a escala irá iniciar na sexta oitava. Para evitar isso, inclua um comando O4 no início da string. Dessa forma a linha se torna:

```
20 PLAY "O4EF+G+ABO5C+D+EF+G+ABO6C+D+E"
```

Cada nota dentro das 8 oitavas disponíveis também pode ser executada usando-se o comando N. Este comando deve ser seguido por um número entre 0 e 96, inclusive. Como existem 96 notas em 8 oitavas, a diferença entre N52 e N53, por exemplo, é de um semitom.

De fato, as 8 oitavas disponíveis no comando N são elevadas de um semitom a partir dos disponíveis no comando O. Isto ocorre pois N0, que corresponderia ao C em O1, é silencioso, e pode ser usado da mesma forma que as pausas. A menor nota disponível, usando esta notação, corresponde a C+ em O1.

A maior nota no comando O é B em O8. Entretanto, N96 é um semitom maior e toca perto do C acima dessa.

Observe que o "C médio" está em N36.

O programa seguinte toca a mesma escala em MI maior, mas desta vez usando o comando N:

```
10 REM Mi Maior:Notacao N
20 PLAY"n40n42n44n45n47n49n51"
30 PLAY"n52n54n56n57n59n61n63n64"
```

Se você sabe ler música, achará a notação A-G mais fácil de usar.

Até agora todas as notas foram executadas com a mesma duração. Para alterar a duração das notas seguintes, use o comando L. Esse comando deve ser seguido por um número que pode assumir qualquer valor entre 1 e 64. Os valores mais úteis são listados abaixo:

Ln	Duração da Nota 1/n.
L1	Toca uma nota inteira ou semibreve.
L2	Toca meia nota ou mínima.
L4	Toca uma nota com um intervalo de um quarto ou semínima.
L8	Toca uma nota com um intervalo de um oitavo ou colcheia.
L16	Toca uma nota com um intervalo de um dezesseis avos ou semicolcheia.
L32	Toca uma nota com um intervalo de um trinta e dois avos ou fusa.

Você também pode usar L3, L5 etc. L3 produziria uma nota com um terço do comprimento de uma nota inteira.

Se você não definir o comprimento das notas, ele será assumido automaticamente como semínima, isto é, L4 é o valor padrão.

Para alterar a escala de E maior, de modo que ela seja tocada em colcheias, você precisará inserir L8 no início de uma string PLAY. O comando L afeta todas as notas seguintes do programa.

O programa seguinte toca MI maior em colcheias, usando a notação A-G. Observe que a duração da nota somente precisa ser definida uma vez no programa:

```
10 REM Mi Maior:Colcheia
20 PLAY"l8ef+g+abo5c+d+e"
30 PLAY"n52n54n56n57n59n61n63n64"
```

Para determinar a duração de uma simples nota, você não deve usar a letra L, mas sim colocar o número, que especifica a duração da nota, após a nota ser tocada:

A + 16 é o equivalente a L16A+

Observe que você não pode fazer isso na notação N.

Para especificar uma pausa, usa-se o comando R. O valor padrão desse comando é R4, que é o equivalente a uma pausa de colcheia. Para alterar a duração da pausa, siga o comando R com um número entre 1 e 64. Esse determina a duração da pausa. Desse modo:

Rn	Duração da pausa = 1/n.
R1	Pausa de uma nota inteira.
R3	Pausa de um terço de uma nota.
R2	Pausa de metade de uma nota.
R4	Pausa de um quarto de uma nota.
R8	Pausa de um oitavo de uma nota.
R16	Pausa de um dezesseis avos de uma nota.
R32	Pausa de um trinta e dois avos de uma nota.
R64	Pausa de um sessenta e quatro avos de uma nota.

Se você quiser tocar uma nota ou pausa "pontuada", siga o comando dessa nota com um ponto ou parada total. Cada ponto após a nota faz com que a duração dela aumente pela metade em relação ao seu valor òriginal. Por exemplo:

```
10 REM notas pontilhadas
20 PLAY"ar64a.r64.."
```

O programa toca a nota como semínima, seguida por uma semínima pontuada e finalmente uma semínima pontuada duas vezes!

As pausas são incluídas de modo que você possa ouvir as notas individualmente. Sem essas pausas você iria ouvir somente uma nota A tocada apenas uma vez.

O tempo de uma peça musical pode ser determinado usando-se o comando T. O número que segue ao comando pode assumir qualquer valor inteiro entre 32 e 255, e determina o número de semínimas tocadas por minuto. Dessa forma, você pode determinar o tempo de uma peça de música, desde as lentas com 32 semínimas por minuto até as vigorosas com 255 semínimas por minuto.

O tempo padrão é T120.

No último exemplo da escala do MI maior, alteramos todas as notas para colcheias, de modo que a escala foi tocada duas vezes mais rapidamente que no exemplo anterior. Um modo alternativo para aumentar a velocidade da execução seria alterar o tempo em T240. As notas ainda são tocadas como semínimas, mas cada semínima é tocada duas vezes mais rapidamente que antes:

```
10 REM Mi Maior:Tempo Fixo
20 PLAY"t240ef+g+abo5c+d+e"
```

O tempo deve ser determinado uma única vez no programa.

O comando V determina o volume. O número que o segue deve estar entre 0 e 15 inclusive. V0 é muito baixo; V15 é muito alto.

O valor padrão é V8.

Da mesma forma que os comandos O, L e T; uma vez definido, esse comando irá permanecer com o valor definido até você redefini-lo ou até que o computador seja desligado.

Se uma frase for repetida freqüentemente em uma peça de música você pode criar uma string contendo a frase. Quando usar a string em um comando PLAY, você deve preceder o nome da string com um comando X e segui-la por ponto e vírgula.

Duas variáveis string, A$ e B$, são usadas no programa-exemplo seguinte. O tempo é armazenado na variável numérica *tempo* de modo que você pode ouvir o efeito dessa alteração. Tente *tempo* de 120 e 240:

```
10 REM MELODIA
20 INPUT"Tempo";TEMPO
30 A$="v8ar64ar64ab-2r64b-o5co4b-aga"
40 B$="b-o5c04fb-a2gf2.f2."
50 GOSUB1000
60 GOSUB1000
70 PLAY"v9o5cr64co4ao5cf2cr64co4ao5cr64c2"
80 PLAY"cd2cr64co4b-ar64a2.g2."
90 GOSUB1000
100 END
990 REM sub-rotina
1000 PLAY"t=tempo;xa$;"
1010 PLAY"xb$;"
1020 RETURN
```

OBSERVAÇÕES

Caso todas suas notas pareçam se encontrar ou não haja separação entre duas notas com o mesmo tom (AA por exemplo), então diminua o número de modulação.

Há apenas oito formas de onda disponíveis. Os números delas são 0, 4, 8, 10, 11, 12, 13 e 14. Qualquer outro número repetirá a mesma forma.

A oitava mais alta soará mais como uma campainha ao passo que a mais baixa soará mais como um tambor.

NOTAS DE ENCERRAMENTO

Depois que você tiver dominado os fundamentos da programação BASIC, conforme mostrado neste livro, poderá voltar ao manual usuário do MSX e melhorar seu conhecimento. O seu manual deverá ser muito fácil de compreender com a introdução à programação BASIC que este livro lhe forneceu.

Não se preocupe se não conseguir aprender imediatamente todos os comandos disponíveis. Você pode escrever programas com apenas alguns dos comandos e melhorar a sua programação e conhecimento à medida que progredir.

Para começar, escreva programas simples e não escreva demais inicialmente. Desmembre seus programas em partes, escreva e teste estas partes antes de seguir adiante. Utilize as declarações REM sempre que possível para comentar os programas que escreve.

Um último conselho é de salvar seus programas periodicamente, mesmo que não estejam prontos. Muitos programadores já arrancaram os cabelos porque seu cão de estimação tropeçou no cabo de alimentação, ou porque um erro no programa perdeu um trabalho de três horas.

Você provavelmente já notou que no final de cada frase as vozes estão ligeiramente fora de sincronia. Isto é devido ao fato de que as strings de cada voz contêm números diferentes de comandos, algumas strings têm mais pausas e mudanças de oitavas que as outras. Cada comando toma um tempo muito pequeno. Você pode tentar sincronizar as strings inserindo umas poucas "R64" e mudanças de oitavas em locais apropriados. (*Nota:* Um comando O4 não fará nada se a string já estiver nessa oitava, mas ela irá incrementar o tempo de execução global da string.)

Até aqui todas as notas que você tocou soaram num tom muito sincronizado. É possível alterar isso colocando uma envoltória no volume das notas; essa envoltória irá controlar a qualidade do som da nota. Existem diversas envoltórias de volumes predefinidas que você pode escolher; veja a Parte 3: Som.

Para selecionar uma envoltória, use o comando S seguido por um número entre 0 e 15. O período de uma envoltória, ou modulação, é determinado pelo comando M. Isto é determinado por qualquer valor entre 1 e 65535; o valor padrão é 255. Digite o seguinte:

```
10  PLAY"S10L1CDE"
```

Você irá ouvir uma envoltória de período de um décimo colocada sobre as notas C, D e E.

Tente alterar o número da envoltória. Digite e execute:

```
10  PLAY"M1000S10L1CDE"
```

Agora, com a mesma envoltória, tente outros valores de M.

Faça experiências com outros valores de M e S. A ordem dos comandos S e M não é importante, mas ambos têm de vir antes das notas a serem executadas.

Finalmente vamos ver o comando BEEP. Este é o comando mais simples de ser usado, e diz ao computador que faça somente um ruído. Digite:

```
BEEP    <RETURN>
```

Ao executar um BEEP, o computador redefine todas as variáveis dos sons. Desse modo, após um BEEP, todos os macrocomandos musicais são redefinidos para os seus valores padrões.

Parte II

PROGRAMAÇÃO

EDIÇÃO AVANÇADA DE PROGRAMAS

O MSX-BASIC tem um editor que permite editar um programa em qualquer parte do modo de tela.

Esta seção entra em detalhes técnicos sobre a edição, bem como a edição avançada do MSX usando a tecla de controle <CTRL>.

COMO EDITAR

Quando se edita, você freqüentemente precisa escrever uma série de linhas, ou cancelar seções inteiras do seu programa. O MSX-BASIC tem quatro comandos muito úteis que podem tornar menor o tédio da edição: LIST, AUTO, RENUM e DELETE.

Uma vez que tenha digitado o seu programa na memória do computador, você irá querer revê-lo antes de executá-lo. Para examinar o programa inteiro use o comando LIST. Ele irá exibir o programa na ordem clara e lógica; o programa pode ser então editado utiizando-se os vários dispositivos de edição do MSX.

O comando LIST tem as seguintes variações que podem ser usadas de acordo com as suas necessidades:

LIST	Lista o programa inteiro.
LIST <linha>	Lista a linha especificada.

LIST <linha1>–<linha2>	Lista o programa entre a linha1 e a linha2 inclusive.
LIST <linha>–	Lista o programa a partir da <linha> especificada até o fim.
LIST –<linha>	Lista o programa a partir do início até a <linha> especificada.

Você observará que, se o programa que você estiver listando for muito longo, o LIST executará o programa no modo de tela até a última linha deste, ou seja, até a última linha especificada no comando LIST.

Existem dois métodos para se exibir uma parte do programa que nos interessa no modo de tela:

a) Use LIST <linha1>–<linha2> de modo que você só obtenha a seção desejada.

b) Use LIST e, quando você vir a parte do programa que lhe interessar, pressione <CTRL><STOP> e interrompa a listagem.

Observe que a tecla <STOP> irá parar apenas temporariamente a listagem. Se você pressionar a tecla <STOP> outra vez, a listagem continuará a partir do ponto onde havia parado.

AUTO

O comando AUTO coloca o MSX no modo de numeração automática de linha, que oferece um número de linha a cada vez que você pressiona a tecla <RETURN>. Ele economiza o aborrecimento de se digitar o número de linha a cada nova linha, e torna a vida um pouco mais fácil para o programador. Os números de linha terão um incremento constante.

O comando AUTO tem as seguintes variações, que podem ser usadas de acordo com as suas necessidades:

AUTO	Fornecerá números de linha a partir do número 10 com incrementos de 10 a cada <RETURN>.
AUTO <linha>	Fornecerá números de linha a partir do número de <linha> especificado com incrementos de 10 a cada <RETURN>.

AUTO <linha>, <incremento>	Fornecerá números de linha a partir do número de <linha> especificado com incrementos de <incremento>.
AUTO, <incremento>	Irá continuar a partir da linha em que você parou, com o <incremento> especificado.
AUTO <linha>,	Fornecerá números de linha a partir do número de linha especificado com o incremento já especificado para as linhas anteriores.

As duas maneiras mais comuns de se usar o comando AUTO, quando se escreve ou edita um programa, são:

1. Quando se inicia um novo programa que já tenha sido planejado de alguma forma no papel, o AUTO é usado para se ter números de linha a partir de 10, com incrementos de 10. Isso dá ao programa uma aparência ordenada.
2. Vamos dizer que você tenha as linhas 100, 110, 120, 130, 140, 150, 160 e queira inserir uma sub-rotina entre as linhas 150 e 160. Antes de mais nada use RENUM para criar mais espaço entre as linhas 150 e 160. Isto pode ser feito com RENUM 1000, 150, 100. Como resultado você terá as linhas 100, 110, 120, 130, 140, 1000, 1100. Use agora AUTO 1010, 5 que lhe dará uma série de números de linha iniciando em 1010, com incremento de 5. O resultado final se parecerá com este:

   100, 110, 120, 130, 140, 150, 1000, <1010, 1015, 1020, 1025, ... >, 1100

Existem alguns pontos a serem lembrados sobre o comando AUTO:

1. Existem duas maneiras de sair do modo AUTO:

   a) Pressione <CTRL> <STOP> ao mesmo tempo.

   b) Pressione <CTRL> <C> ao mesmo tempo.

   A linha da qual você saiu não será armazenada.

2. Se existir um número de linha dentro do programa que você está editando, o computador dará a você um sinal de atenção na forma de asterisco "*" após o número da linha. Se você não quiser apagar a linha, pressione a tecla <RETURN>, mas se estiver substituindo a linha, ignore o aviso "*" e digite o que você quiser.

RENUM

Tendo digitado o seu programa, invariavelmente você descobrirá que os números de linha não estão a incrementos fixos. Ele parece confuso e, se o programa tiver de ser apresentado a uma outra pessoa, certamente dará uma péssima impressão. A cura é simples: renumere o programa inteiro, usando o comando RENUM.

RENUM, por si só, irá renumerar a começar do 10, em incrementos de 10. Essa é a forma mais usada, mas você pode renumerar a partir de qualquer número de linha e com vários incrementos. RENUM irá alterar automaticamente os números de linha associados a GOTO, GOSUB, RESTORE, THEN, ELSE, ON GOTO, ON GOSUB.

O comando RENUM tem as seguintes variações que podem ser usadas de acordo com as suas necessidades:

RENUM	Renumera a partir da linha 10, em incrementos de 10.
RENUM <linha1>	Renumera a partir da <linha1>, em incrementos de 10.
RENUM <linha1>, <linha2>	Renumera a partir da antiga <linha2>, iniciando com a nova <linha1> em incrementos de 10.
RENUM <linha1>, <linha2>, <incremento>	Renumera a partir da antiga <linha2> iniciando com a nova <linha1> com o <incremento> informado.
RENUM ,<linha2>, <incremento>	Renumera a partir da antiga <linha2> iniciando na linha 10 com o <incremento> informado.
RENUM ,<linha2>	Renumera a partir da antiga <linha2> iniciando na linha 10 com o incremento de 10.
RENUM ,, <incremento>	Renumera a partir da linha 10 com o <incremento> informado.
RENUM <linha1>,, <incremento>	Renumera iniciando com a nova linha <número1> a partir da antiga linha 10 com o <incremento> informado.

RENUM é também usado para criar espaços de números de linha suficientes para você inserir uma parte em um programa (como quando foi usado com AUTO, como mostrado anteriormente).

DELETE

Algumas vezes você vai achar que certa parte de um programa é um lixo total. Se a parte que você quiser apagar for uma única linha, então você só precisa digitar o número de linha e pressionar <RETURN>. Entretanto, se você quiser apagar uma série inteira de linhas, use o comando DELETE. É simples e efetivo. É também permanente, de modo que, uma vez apagadas, não há meio de recuperar essas linhas, exceto digitando-as de novo!

O comando DELETE tem as seguintes variações que podem ser usadas de acordo com as suas necessidades:

DELETE <linha>	Apaga a <linha> especificada.
DELETE <linha1> – <linha2>	Apaga entre a <linha1> e <linha2>.
DELETE – <linha2>	Apaga até a <linha2> inclusive.

TECLAS DE CONTROLE E DE FUNÇÕES ESPECIAIS

O MSX tem um certo número de teclas de funções especiais e de controle usadas principalmente na edição de programas. Eis uma tabela para mostrar o que você pode fazer com elas e como acessá-las. Todas as teclas de controle são acessadas pressionando-se <CTRL> e a tecla em questão, simultaneamente.

Código Hexadecimal	Tecla de Controle	Tecla Especial	Função
02 *	<CTRL><B>		Move o cursor uma palavra para a esquerda.
03 *	<CTRL><C>		Pára a execução quando o BASIC estiver esperando por INPUT. Retorna para o modo direto.
05 *	<CTRL><E>		Apaga todo o texto à direita do cursor, incluindo o caractere na posição do cursor.
06 *	<CTRL><F>		Move o cursor uma palavra para a direita.

* Indica que desativará o modo de INSERT se estiver ativo.

Código Hexadecimal	**Tecla de Controle**	**Tecla Especial**	**Função**
07 *	<CTRL><G>		Toca o bip.
08 *	<CTRL><H>	BS	Retorna um espaço e apaga o caractere à esquerda do cursor.
09	<CTRL><I>	TAB	Move o cursor para a posição de tabulação seguinte. TAB é definido para intervalos de 8 caracteres. Ele irá colocar brancos por onde se mover.
0A *	<CTRL><J>		Alimentação de linha. Move o cursor para o início da linha seguinte.
0B *	<CTRL><K>	HOME	Move o cursor para o canto superior esquerdo da tela.
0C *	<CTRL><L>	CLS	Limpa a tela e move o cursor para a posição HOME.
0D *	<CTRL><M>	RETURN	Retorno de carro. Executa a linha atual.
0E *	<CTRL><N>		Move o cursor para o final da linha atual.
12 *	<CTRL><R>	INS	Entra no modo INSERT. O cursor fica com a metade do tamanho e permite inserir caracteres na posição do cursor sem apagar nada.
15 *	<CTRL><U>		Apaga a linha atual.
1B *	<CTRL><< >	ESC	Ignorada nesta versão de MSX.
1C *	<CTRL><Y>	→	Move o cursor para a direita uma posição por vez.
1D *	<CTRL>< >>	←	Move o cursor para a esquerda uma posição por vez.
1E *	<CTRL><^>	↑	Move o cursor para cima uma linha por vez.
1F *	<CTRL><–>	↓	Move o cursor para baixo uma linha por vez.
7F	<CTRL><DEL>	DEL	Apaga o caractere na posição do cursor.

* Indica que desativará o modo INSERT se estiver ativo.

CAPÍTULO 30

CONSTANTES E VARIÁVEIS

CONSTANTES

Constantes são números ou strings que não variam. Por exemplo: 100 e "VELOCIDADE" são constantes.

Existem dois tipos de constantes – strings e numéricas.

Constantes strings

Uma constante string é uma seqüência de até 255 caracteres de comprimento. Elas podem conter quaisquer caracteres, gráficos ou códigos de controle, que são usados no MSX-BASIC. Elas devem vir entre aspas:

```
"LUKE SKYWALKER"
"O QUE É ISSO?"
"ANTONIO, RICARDO E HENRIQUE"
"Cz$100.000.000,00"
```

Constantes numéricas

Elas podem ser números positivos ou negativos e são de seis tipos:

1. Constantes inteiras Números inteiros entre –32768 e 32767 ou de

	1111111111111111 a 0000000000000000 em 16 bits. Elas não podem conter pontos decimais.
2. Constantes de ponto fixo	Números reais positivos ou negativos que algumas vezes contêm ponto decimal, isto é: 657.188.
3. Constantes de ponto flutuante	Números reais negativos ou positivos na forma exponencial. Formatos: Precisão simples <inteiro> E<inteiro> –56E10 <ponto fixo> E<inteiro> 7.865E5 Precisão dupla <inteiro> D<inteiro> –1896243232662D50 <ponto fixo> D<inteiro> 7.8827727277275D-5 O limite de precisão está entre 1E-64 e 1E64 (1×10^{-64} a 10^{64}).
4. Constantes hexadecimais	Números hexadecimais denotados pelo prefixo &H de precisão simples ou dupla.
5. Constantes em octal	São representados pelo prefixo &O, e trabalha com dígitos de 0 a 7. Exemplo: &O6543.
6. Constantes binárias	Números binários denotados pelo prefixo &B. Binário é um sistema numérico de base 2, isto é, usa 0 e 1 somente. Exemplo: &B01010101.

Precisão simples e precisão dupla

Uma das características principais do MSX-BASIC é que ele tem funções aritméticas BDC com 14 dígitos de precisão dupla, o que a maioria dos computadores de 8 bits não possui.

A precisão dupla é normalmente oferecida em sistemas de 16 e 32 bits mas a Microsoft reescreveu o seu BASIC para fornecer cálculos dessa precisão.

Você pode ter constantes numéricas de precisão simples ou dupla mas, se não especificar qual, o computador assumirá precisão dupla.

Precisão:	Precisão simples: 6 dígitos.
	Precisão dupla: 14 dígitos.

As constantes de precisão simples têm as seguintes características:

1. Forma exponencial (E)	1.65E−2
2. Ponto de exclamação no final	235.77!

As constantes de precisão dupla têm as seguintes características:

1. Qualquer dígito ou número sem um ponto de exclamação ou exponencial	1878932.2	156
2. Exponencial usando D	562.765333D−06	
3. Sinal # no final	387.0001 #	

VARIÁVEIS

Variáveis são nomes aos quais você atribui valores. Os valores de uma variável podem ser alterados. Eles podem ser atribuídos especificamente por um programador (isto é, PI=3.14) ou calculadas pelo computador (isto é, PI=4*ATN(1)).

Nomes de variáveis

Os nomes de variáveis podem ter qualquer tamanho. Entretanto, o computador somente reconhece os primeiros dois caracteres de qualquer nome de variável. Se duas variáveis tiverem as mesmas duas primeiras letras, elas são assumidas como sendo a mesma (VAR1 e VAR2 são a mesma variável). Se você quiser fazê-las diferentes, use V1 e V2.

Os nomes de variáveis não podem conter palavras reservadas, ou seja, comandos e funções do BASIC em qualquer parte do nome. Por exemplo, LENTE% está errado pois contém o comando LEN.

Declaração de tipos

Se uma variável não contiver qualquer tipo de caractere de declaração seguindo o seu nome, então ela é assumida pelo computador como sendo uma variável numérica de precisão dupla.

A1 = <número de precisão dupla>

Sufixos de declaração de tipos:

$ Um sinal de cifrão indica que a variável é string:

Z$="string de caracteres".
ENDER$="Rua do Padeiro, 21B".

% Um sinal de percentagem indica uma variável inteira:

X%=25
XY%=32768

! Um ponto de exclamação indica uma variável numérica de precisão simples:

F!=5679.7!
PRESI!=4,2888E–02

O sinal # indica uma variável numérica de precisão dupla:

H # =3.14161718293
MXM # =5277.7655D25

DEF <tipo de variável>

É possível definir um tipo de variável usando vários comandos DEF. Quando o tipo da variável é declarado usando um desses comandos, qualquer variável iniciando com o <caractere> declarado se tornará um tipo de variável que não necessita usar sufixos em seu nome. Isto significa que se você define DEFSTR A, então qualquer variável que se inicie com a letra A se tornará uma variável string mesmo sem ter o sinal de cifrão usual. Entretanto, isso não significa que AB% será uma variável string, já que o caractere de declaração de inteiro (%) tem prioridade sobre DFSTR.

Existem quatro comandos DEF <tipo de variável>. São os seguintes:

- DEFSTR declara que qualquer nome de variável que se inicie com a faixa de caracteres especificada será tratada como uma variável string:

```
DEFSTR A
AST="PLANETA"
```

- DEFINT declara que qualquer nome de variável que se inicie com um caractere na faixa de caracteres especificada será tratada como uma variável inteira.

- DEFSNG declara que qualquer nome de variável que se inicie na faixa especificada de caracteres será tratada como um número de precisão simples.

- DEFDBL declara que qualquer nome de variável que se inicie na faixa de caracteres especificada será tratado como um número de precisão dupla.

Observe que os caracteres #, $, % e ! têm prioridade sobre esses comandos.

MATRIZES

Uma matriz é um grupo de variáveis que possui um nome comum. Ela é definida usando-se o comando DIM. Comandos DIM definem uma seção de memória à parte para o uso da matriz. Cada elemento da matriz tem um número. Por exemplo, digamos que você definiu DIM C(3). Você tem, então, quatro variáveis com o nome C( ): C(0), C(1), C(2) e C(3). Todas as variáveis de matriz iniciam no elemento zero. Esta é uma matriz unidimensional, mas não há razões para você não ter matrizes multidimensionais como DIM A(2,2), que fornece uma tabela de variáveis de dimensão 3 × 3 como se mostra abaixo:

```
A(0,0)A(1,0)A(2,0)
A(0,1)A(1,1)A(2,1)
A(0,2)A(1,2)A(2,2)
```

Esta é uma matriz de duas dimensões, mas você pode ter matrizes de três dimensões ou mais. A dimensão máxima para uma matriz é 255, porém é provável que não exista memória suficiente para criar tal tipo de matriz.

Se uma matriz tiver menos que 12 elementos, não há necessidade de se executar um comando DIM, pois o computador define automaticamente espaços de memória de 11 elementos. Programas como os mostrados a seguir são perfeitamente válidos:

```
10  FOR I=0 TO 10
20  S(I)=1
30  NEXT I
```

É possível ter matrizes de qualquer tipo assim que o tipo for declarado. Por exemplo, DIM S%(100) fornece a você uma matriz string de cento e um elementos.

Quando uma matriz é inicializada, todos os valores são assumidos como sendo zero para matrizes numéricas e strings nulas para matrizes strings.

Requisitos de memória

As variáveis são armazenadas na ÁREA DE MATRIZES, na ÁREA DE VARIÁVEIS ou na ÁREA DE STRINGS, dependendo do seu tipo (veja o mapa de memória).

Eis uma lista do número de bytes usados por essas variáveis:

Variáveis	Inteira	2 bytes
	Precisão simples	4 bytes
	Precisão dupla	8 bytes
	String	3 mais o tamanho do conteúdo
Matrizes	Inteira	2 bytes por elemento
	Precisão simples	4 bytes por elemento
	Precisão dupla	8 bytes por elemento
	String	3 mais o tamanho do conteúdo por elemento

VARPTR

Para descobrir onde exatamente os dados de uma variável serão armazenados na

memória, você deve usar a função especial VARPTR que fornece o endereço de uma variável em particular:

```
10  A%=100
20  PRINT VARPTR (A%)
```

Você pode descobrir a localização de memória de qualquer tipo de variável desde que ela exista. Ela pode ser uma string ou mesmo uma matriz.

Área de variável string

O conteúdo das variáveis string é armazenados na ÁREA DE STRINGS da memória do MSX (veja o mapa de memória). O tamanho dessa área está restrito ao padrão inicial de 200 bytes; desse modo, o tamanho máximo de uma string está limitado a 200 caracteres a qualquer tempo. Entretanto, você pode aumentar o tamanho da ÁREA DE STRINGS usando o comando CLEAR. Por exemplo, para aumentar o tamanho da ÁREA DE STRINGS para 255 bytes, execute:

```
CLEAR 255
```

CAPÍTULO 31

CONVERSÃO DE TIPOS DE VARIÁVEIS

O MSX-BASIC pode converter um tipo de constante numérica em outro. Existe uma regra definida para se fazer isso:

1. Se uma constante numérica de um tipo é definida igual a outra de outro tipo, o número armazenado irá depender inteiramente do tipo da variável.

Exemplo 1

```
10 C!=1.2345678901
20 PRINT C!
run
 1.23457
```

O número de precisão dupla 1.234567890 é convertido para uma variável de precisão simples quando atribuído a C!.

Exemplo 2

```
10 I%=1.23E-03
20 PRINT I%
run
 0
```

O número de precisão simples 1.23E–03 é arredondado para o inteiro mais próximo, que é 0, quando atribuído a uma variável inteira.

2. Durante a execução da expressão, todas as variáveis e constantes são convertidas para a maior precisão das variáveis usadas. Quando a expressão é definida para ser igual a uma variável, o resultado da expressão é convertido para o tipo de precisão especificado pela variável. Se o tipo de variável resultante não for especificado, o resultado permanece com o mesmo nível de precisão que tinha durante a execução.

Exemplo 1

```
10 D=1!/3#
20 PRINT D
run
 .3333333333333
```

1! tem precisão simples enquanto 3# tem pecisão dupla. A operação artitmética foi executada com precisão dupla e armazenada na variável D de precisão dupla.

Exemplo 2

```
10 D=1!/3!
20 PRINT D
run
 .333333
```

Ambas as constantes são de precisão simples, de modo que as operações aritméticas são executadas com precisão simples, mas D é uma variável de precisão dupla, de modo que o resultado é convertido para precisão dupla.

Exemplo 3

```
10 D!=1/3
20 PRINT D!
run
 .333333
```

As operações aritméticas são executadas com precisão dupla, mas o resultado é convertido para precisão simples quando armazenado em D!.

3. Operadores lógicos convertem as constantes e variáveis em números inteiros de 16 bits antes da operação ser executada. O resultado é um inteiro e os números envolvidos devem estar dentro da faixa de inteiros ou irá ocorrer um erro.

Exemplo

```
10 D%=156.65 AND 654
20 PRINT D%
run
 140
```

Os números são alterados para 156 e 654. O resultado 140 é armazenado em D% que é uma variável inteira.

4. Quando um valor de ponto flutuante ou de ponto fixo é convertido para inteiro, a parte fracionária é truncada.

Exemplo 1

```
10 A%=123.456
20 PRINT A%
run
 123
```

O valor de ponto fixo 123.456 é truncado para 123 quando atribuído à variável inteira A%.

Exemplo 2

```
10 A%=1.0097554D02
20 PRINT A%
run
 100
```

O número de ponto flutuante 1.00097554D02 é arredondado para 100 quando convertido para A%, uma variável inteira.

5. Quando um número de precisão simples é convertido para um número de precisão dupla, somente 6 dígitos podem ser fornecidos para o resultado de precisão dupla.

Exemplo

```
10 A#=1.33333
20 PRINT A#
run
 1.33333
```

A# é uma variável de precisão dupla mas o resultado é 1.33333.

CONVERSÃO DE TIPOS USANDO AS FUNÇÕES DO MSX-BASIC

CDBL, CINT, CSNG, VAL, STR$

Apesar do MSX fazer a conversão de tipos automaticamente quando está calculando uma expressão, você pode programá-lo para fazer conversão de tipos usando uma das funções de conversão de tipo.

CDBL, CINT e CSNG seguem as mesmas regras acima, mas as funções VAL e STR$ são usadas para converter de string para numéricas e vice-versa, sendo que elas seguem regras ligeiramente diferentes.

CINT

A função CINT converte números de precisão simples, precisão dupla, variáveis e expressões em números inteiros. O argumento deve estar estritamente dentro da faixa de inteiros (de –32768 a 32767). Caso contrário ocorrerá um erro.

```
CINT(<constante numérica>)
CINT (<variável numérica>)
CINT (<expressão numérica>)
```

Exemplo:

PRINT CINT(1234.56789)
1234

CSNG

A função CSNG converte o argumento em um número de precisão simples. CSNG arredonda o argumento para a sexta casa decimal mais próxima. Se um inteiro for convertido, a precisão não será alterada.

CNSG(<constante numérica>)
CNGS (<variável numérica>)
CNGS (<expressão numérica>)

Exemplo:

PRINT COS(0.7656)
.72096670541357
PRINT CSNG(COS(0.7656))
.720967

CDBL

A função CDBL converte números inteiros, números de precisão simples, variáveis e expressões em números de precisão dupla. Quando um número de precisão dupla é convertido, somente 6 dígitos serão fornecidos no resultado de precisão dupla.
dupla.

CDBL (<constante numérica>)
CDBL (<variável numérica>)
CDBL (<expressão numérica>)

Exemplo:

PRINT CDBL (5.666!)
5.666

VAL e STR$

Essas funções são usadas para converter strings em números e vice-versa.

STR$ (<numérico>)

STR$ converte o argumento numérico em uma string.

VAL (<string>)

VAL retorna o valor de uma string que contém um número. Se uma string for uma representação da string de um número, então pode ser usada VAL. Entretanto, VAL não pode executar uma expressão contida numa string.

VAL elimina espaços, tabulações e alimentação de linhas da frente de um número em uma string, mas não pode trabalhar com outros caracteres. VAL reconhece sinais de mais e de menos.

Veja o Capítulo 15 para exemplos.

CAPÍTULO 32

EXPRESSÕES E OPERADORES

Uma expressão pode ser uma string, uma constante numérica ou uma variável, ou qualquer tipo de combinação que retorne um valor simples, desde que válido. Existem quatro tipos de operadores matemáticos ou lógicos:

1. Aritmético
2. Relacional
3. Booleano (lógico)
4. Funcional

Os tópicos **1** e **2** são cobertos nesta parte do livro (veja as seções relevantes que são relacionadas com os outros operadores).

OPERADORES ARITMÉTICOS

Os operadores aritméticos, em ordem de precedência, são:

1. ^	exponencial	A^B
2. –	negação	–A

3. */ multiplicação e divisão A*B, A/B

4. +,– adição e subtração A+B, A–B

Para alterar a ordem dos cálculos use parênteses ( ).

A ordem de precedência é mantida dentro dos parênteses.

DIV e MOD (divisão de inteiro)

DIV é denotado pelo sinal \ (barra invertida) no MSX-BASIC. Ele executa uma divisão de inteiro. O resto desta divisão é dado pelo MOD.

O que são DIV e MOD? Eis um exemplo para ilustrar seus usos. Numa divisão normal, se você dividir 10 por 4, irá obter 2.5.

De maneira similar,

13/5 = 2.6

Entretanto, em uma divisão de inteiro você não obtém o ponto decimal, mas em vez disso obtém a parte inteira do resultado da divisão, 2 e um resto, 3.

No MSX-BASIC, se você executar 13 DIV 5 obterá:

13\5=2...a parte inteira do número.

Se você executar 13 MOD 5=3...o resto da divisão.

As operações aritméticas divisão de inteiro e resto são executadas com 16 bits. Antes do cálculo ser feito, qualquer não inteiro é convertido para inteiro por meio de truncamento, isto é, todas as partes fracionárias são descartadas.

16.78\5.89 = 3 (é o mesmo que 16\5 = 3)
16.78 MOD 5.89 = 1 (é o mesmo que 16 MOD 5 = 1)

MOD segue DIV (\) na ordem de precedência.

Exemplo: Mostrando a ordem de precedência de MOD e DIV:

1580 MOD 789 DIV10

1. 789 DIV 10
78

2. 1580 MOD 78
. 20

isto é, 1580 MOD 789\10 = 20.

OPERADORES RELACIONAIS

Operadores relacionais comparam dois valores e retornam uma resposta verdadeira (–1) ou falsa (0). Eles são normalmente usados em condições IF THEN, mas podem ser incorporados em uma expressão.

OPERADOR RELACIONAL		EXEMPLO
=	Igual a	A=B
< >	Diferente de	A< >B
<	Menor que	A < B
>	Maior que	A > B
< =	Menor ou igual a	A< =B
> =	Maior ou igual a	A> =B

Verdadeiro retorna –1. Falso retorna 0.

Quando existir uma mistura de operadores relacionais e aritméticos, os últimos terão precedência.

Exemplo

A*B = C

A*B é executado primeiro, e então o resultado é comparado com C.

Operadores relacionais são tratados em detalhes na seção IF/THEN/ELSE (Capítulo 35).

CAPÍTULO 33

INTRODUÇÃO AOS SISTEMAS NUMÉRICOS USADOS NO MSX

Como a maioria dos seres humanos deste planeta tem dez dedos em suas mãos, eles usam o sistema de numeração decimal. Ao contrário do *homo sapiens*, os computadores, que são basicamente uma imensa matriz de chaves, podem ter somente dois "dedos" – LIGADO e DESLIGADO (ou, em outras palavras, 0 e 1). Desse modo, os computadores trabalham em binário, internamente, o tempo todo. Para conforto dos usuários, eles exibem os números em decimal mas, naturalmente, eles também podem trabalhar com números binários em programas de usuários. Além do sistema binário, o MSX pode usar o sistema octal e o hexadecimal. Tanto os hexadecimais como os octais e binários são números inteiros e devem estar dentro da faixa de inteiros, isto é, entre –32768 e 32767.

As funções aritméticas são calculadas no sistema Binário Codificado em Decimal (BCD), que será explicado mais adiante.

BINÁRIO

Definição:	Sistema numérico de base 2. Usa o 0 e o 1 para representar todos os números.
MSX-BASIC	&B <binário> representa um número binário. BIN$ muda de decimal para binário.

Cada dígito de um número binário é chamado "bit". 8 bits são chamados de um "byte". Um byte é o tamanho de um endereço de memória no sistema MSX. Um byte pode conter qualquer valor entre 00000000 a 11111111, ou de 0 a 255 em decimal.

DECIMAL	BINÁRIO	EM 8 BITS
0	0	00000000
1	1	00000001
2	10	00000010
3	11	00000011
4	100	00000100
5	101	00000101
6	110	00000110
7	111	00000111
8	1000	00001000
9	1001	00001001
10	1010	00001010
11	1011	00001011
12	1100	00001100
13	1101	00001101
14	1110	00001110
15	1111	00001111
16	10000	00010000

Conversão de binário para decimal e de decimal para binário

Como converter números binários de 8 bits para o seu decimal equivalente.

NÚMERO DO BIT	CORRESPONDE A	VALOR
7	2^7	128
6	2^6	64
5	2^5	32
4	2^4	16
3	2^3	8
2	2^2	4
1	2^1	2
0	2^0	1

Exemplo: Converter 10110001 para decimal.

```
binário        decimal
&B10110001 = 177
```

Como converter números decimais em binários.

Exemplo: Converter 53 em binário.

BIT	NÚMERO			
7	2^7	128		0
6	2^6	64		0
5	2^5	32	53–32=21	1
4	2^4	16	21–16=5	1
3	2^3	8		0
2	2^2	4	5–4=1	1
1	2^1	2		0
0	2^0	1	1–1=0	1

Arranje os bits e você terá 53 = &B00110101.

Em BASIC:

Como converter um número binário de 8 bits em seu decimal equivalente.

Exemplo:

```
10 X%=&B00101011
20 PRINT X%
run
 43
```

Como converter um número decimal em seu equivalente binário.

```
10 X%=171
20 PRINT BIN$(X%)
run
10101011
```

OCTAL

Definição: Um sistema numérico na base 8. Ele usa números entre 0 e 7 para representar todos os números.

MSX-BASIC: &0 <octal> representa um número octal. OCT$ de decimal para octal.

DECIMAL	OCTAL
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	10
9	11
10	12
11	13
12	14
13	15
14	16
15	17
16	20

Conversão decimal para octal e octal para decimal

Como converter um número octal para o seu decimal equivalente:

<decimal> = <1º dígito> *8^0+ <2º dígito>
*8^1+ <3º dígito> *8^2+ <4º dígito> *8^3...

Exemplo: Octal &O5436 para decimal.

6*8^0+3*8^2+5*8^3=2846
&O5436=2846

Como converter números decimais para octais.

Exemplo: Decimal 6588 para octal:

DÍGITO	DECIMAL		DIVISÃO INTEIRO	MÓDULO OCTAL	
4	8^4	4096	6588 DIV 4096 =	1	6588 MOD 4096 = 2492
3	8^3	512	2492 DIV 512 =	4	2492 MOD 512 = 444
2	8^2	64	444 DIV 64 =	6	444 MOD 64 = 60
1	8^1	8	60 DIV 8 =	7	60 MOD 8 = 4
0	8^0	8	4 DIV 1 =	4	

De modo que você obtém &O014674 = 6588 (decimal).

Em BASIC

Como converter um número octal para o seu decimal equivalente.

Exemplo:

```
10 X%=&O6517
20 PRINT X%
run
 3407
```

Como converter um número decimal para o seu equivalente octal.

Exemplo:

```
10 X%=171
20 PRINT OCT$(X%)
run
253
```

HEXADECIMAL

Definição: Um sistema numérico com base 16. Ele usa números entre 0 e 9 e adicionalmente A, B, C, D, E e F para representar todos os números hexadecimais.

Em BASIC: &H <hex> representa um número hexadecimal
HEX$ converte de decimal para hexadecimal.

DECIMAL	HEXADECIMAL
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	A
11	B
12	C
13	D
14	E
15	F
16	10

Conversão de hexadecimal em decimal e de decimal em hexadecimal

Como converter um número hexadecimal para o seu equivalente decimal:

<decimal> = <prim.dígito> *16^0+ <seg.dígito> *16^1+ <terc.dígito> *6^2+ <quar.dígito> *16^3...

Exemplo: &HFF para decimal.

(&HF)*16^0+(&HF)*16^1=(15)*1+(15)*16=255

&HFF=255

Como converter números decimais para os seus equivalentes hexadecimais.

Exemplo: Decimal 7752 para hexadecimal.

DÍGITO DECIMAL			DIVISÃO INTEIRO	HEXADE-CIMAL	MÓDULO
3	16^3	4096	7752 DIV 4096 =	1	7752 MOD 4096 = 3656
2	16^2	256	3656 DIV 256 =	E(14)	3656 MOD 256 = 72
1	16^1	16	72 DIV 16 =	4	72 MOD 16 = 8
0	16^0	1	8 DIV 1 =	8	

De modo que você obtém &H1E48 = 7752 (decimal).

Em BASIC

Como converter um número hexadecimal no seu equivalente decimal.

Exemplo:

```
10 X%=&HEFA3
20 PRINT X%
run
-4189
```

Como converter um número decimal no seu equivalente hexadecimal.

Exemplo:

```
10 X%=171
20 PRINT HEX$(X%)
run
AB
```

Notas:

1. Binários são freqüentemente usados quando se projeta um sprite porque torna fácil de se ver o padrão dos bits.
2. Hexadecimal é usado na descrição de localizações de memória e em programas em inguagem de máquina.

RESUMO DE DECIMAL, BINÁRIO, OCTAL E HEXADECIMAL

DECIMAL	BINÁRIO	OCTAL	HEXADECIMAL
0	0	0	0
1	1	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F
16	10000	20	10

SISTEMA DECIMAL CODIFICADO EM BINÁRIO

O MSX trabalha com funções aritméticas com precisão dupla de até 14 dígitos. Isto significa que as operações aritméticas não geram os erros de arredondamento que estão invariavelmente associados a muitos sistemas de computadores de 8 bits. Todas as operações e funções aritméticas são calculadas com essa precisão a menos que se especifique outra precisão pelo programa do usuário.

Por que usar decimal codificado em binário? Qual é a vantagem e por que é mais preciso? Antes de iniciar, vamos ver como trabalham os números no sistema binário, que é o mais adequado para os computadores.

Em binário, se você tentar representar um número menor que 1 (isto é, um número com ponto decimal), muitas vezes incorrerá em erros. Existem certos números que podem ser expressos em decimal mas não em binário.

Digamos que queremos converter 0.6525 para binário:

```
0.625 = 1/4+1/8=1/(2^2)+1/(2^3)
0.625 = 2^(-2)+2^(-3)
```

De modo que você pode converter 0.625 para binário sem qualquer problema. Entretanto, quando se tem um número parecido com 0.1, você não pode fazer isso. Não existe modo de se expressar 0.1 em binário de forma exata. A seguir mostramos por que:

```
0.1 ≅ 1/16+1/32+1/256+1/512+1/4096+1/8192+1/65536... ≅ 0.09999...
```

ou

```
0.1 ≅ 1/(2^4)+1/(2^5)+1/(2^8)+1/(2^9)+1/(2^13)+1/(2^16)+... ≅ 0.099999...
```

Você não consegue obter 0.1 em binário; em vez disso você obtém uma série.

Em vez de transformar um decimal completamente em binário, o MSX usa o decimal codificado em binário.

O sistema decimal codificado em binário usa 4 bits binários para representar um dígito em decimal. Isso significa que 0001 corresponde a 1 e 0010 corresponde a 2, e assim por diante até 1001 que é o 9. Os 4 bits binários entre 11 e 15, isto é, 1010, 1011, 1100, 1101 e 1110 e 1111 não são usados no sistema decimal codificado em binário. Em

vez disso, a casa decimal (10) é representada nos próximos 4 bits – assim, 10 é 0001 0000 em BCD. 1 byte (8 bits) representam dois dígitos decimais.

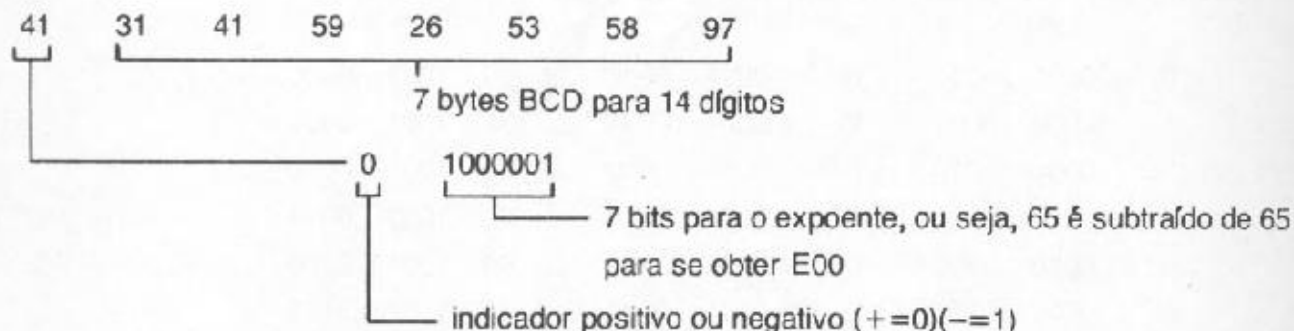
0	0000	0000	10	0001	0000	...	90	1001	0000
1	0000	0001	11	0001	0001	...	91	1001	0001
2	0000	0010	12	0001	0010	...	92	1001	0010
3	0000	0011	13	0001	0011	...	93	1001	0011
4	0000	0100	14	0001	0100	...	94	1001	0100
5	0000	0101	15	0001	0101	...	95	1001	0101
6	0000	0110	16	0001	0110	...	96	1001	0110
7	0000	0111	17	0001	0111	...	97	1001	0111
8	0000	1000	18	0001	1000	...	98	1001	1000
9	0000	1001	19	0001	1001	...	99	1001	1001

Números de 8 bits com precisão dupla em BCD

O MSX armazena dados numéricos em precisão dupla, a menos que seja programado de outra forma. Ele tem uma precisão de 14 dígitos e é armazenado em 8 bytes no sistema decimal codificado em binário. O primeiro byte armazena o expoente e os outros bytes dão os 14 bytes, ou seja, 2 dígitos por byte.

Para ver como os dados numéricos são armazenados, em primeiro lugar pesquise a localização desse dado usando a função VARPTR, e então "espie" (PEEK) o conteúdo e o imprima em hexadecimal.

```
10 PI=3.141592653489#
20 DIREC=VARPTR(PI)
30 FOR I=DIREC TO DIREC+7
40 PRINT HEX$(PEEK(I));" - ";
50 NEXT I
60 PRINT
run
```

Vamos analisar o que foi exposto:

O byte da ponta fornece o sinal positivo ou negativo e o expoente. Para obter o valor exponencial, primeiro ignore os 7 bits e calcule o valor decimal para o resto, isto é, do sexto ao primeiro bit; então subtraia 65. Isto fornece um número com tamanho entre 1E – 64 e 9,9999999999999E62.

VANTAGENS E DESVANTAGENS DO SISTEMA DECIMAL CODIFICADO EM BINÁRIO

Vantagens:

1. O BCD não produz erros de arredondamento e pode fornecer números que são impossíveis em binário como 0.1.
2. É extremamente fácil exibir um número BCD porque não é necessário converter binário para decimal. Isso aumenta a velocidade do processo de impressão na tela.

Desvantagem:

1. Devido ao fato de não ser realmente binário, o computador gasta mais tempo na computação de operações aritméticas. Entretanto, a CPU Z80 tem um comando especial DAA (Decimal Adjust Accumulator = Acumulador de Ajuste Decimal), para tornar esse processo mais fácil. O comando DAA converte o conteúdo do acumulador para BCD compactado seguindo à subtração ou adição, com operandos de compactação BCD.

CAPÍTULO 34

ÁLGEBRA BOOLEANA I (EXPRESSÕES LÓGICAS)

INTRODUÇÃO

Por menor que pareça o seu MSX, ele ainda é um computador, e um computador é basicamente feito de milhões de chaves. Elas podem ser LIGADAS ou DESLIGADAS, mas combinações delas criam um computador que trabalha. A lógica de um computador, ou uma imensa penca de chaves se você quiser, é governada por um conjunto de regras que são conhecidas como Álgebra Booleana. Ela é usada principalmente em situações de múltiplas condições, em comandos IF/THEN, com ANDs e ORs; você também pode usá-la em programas em linguagem de máquina.

OPERADORES LÓGICOS DO MSX

O MSX fornece Operadores Booleanos (lógicos), apesar de alguns micros de 8 bits parecerem tê-los esquecido. Eles são muito importantes em programação avançada. Eis uma lista dos mesmos em ordem de precedência.

		Jargão de computação
1	NOT	Complemento
2	AND	AND lógico

3	OR	OR lógico
4	XOR	OR exclusivo
5	EQV	Equivalente
6	IMP	Implicação

NOT, AND e OR são operadores importantes, já que eles são os operadores lógicos mais comumente usados; e, por outro lado, os outros, isto é, XOR, EQV e IMP, podem ser derivados de combinações dos operadores NOT, AND e OR.

Operações lógicas são sempre executadas com números inteiros. Qualquer argumento que não seja um inteiro é convertido para 16 bits, com sinal, com complemento de dois inteiros.

Agora iremos ver como eles funcionam e para que eles podem ser usados, um a um, iniciando com NOT.

NOT

NOT é uma negação. Ela fornece verdadeiro para falso e falso para verdadeiro (por exemplo: 1 para 0 e 0 para 1). Em outras palavras, ela nega um argumento.

Eis a tabela da verdade para NOT.

NOT	X	NOT X
	0	1
	1	0

NOT X	100	0000000001100100
	−101	1111111110011011

AND

As operações algébricas booleanas de AND podem ser listadas em uma tabela da verdade como mostrada abaixo:

AND	X	Y	X AND Y
	0	0	0
	1	0	0
	0	1	0
	1	1	1

Dessa forma, digamos que 50 AND podem ser calculadas como a seguir:

X	100	0000000001100100
AND Y	50	0000000000110010
	32	0000000000100000

Exemplo: Uso do AND

O AND lógico pode ser usado para testar um bit de um número em particular. Por exemplo, se y = 2^N, onde n é o bit de teste entre 0 e 7, então X AND Y = Y se o bit de teste n for 1 e X AND Y = 0 se o bit de teste n for 0.

TESTE o quinto BIT (n=5) para X=117.

Y = 2^5 = 32

X	117	0000000001110101
AND Y	32	0000000000010000
	32	0000000000010000

Assim, o quinto bit do número 117 é 1, isto é, verdadeiro.

OR

Quando ou X ou Y ou ambos forem verdadeiros, OR irá retornar verdadeiro. O OR lógico é algumas vezes chamado soma lógica.

A tabela da verdade do OR é:

OR	X	Y	X OR Y
	0	0	0
	0	1	1
	1	0	1
	1	1	1

Exemplo: 34 OR 67

X	34	0000000000100010
OR Y	67	0000000001000011
	99	0000000001100011

Exemplo: Uso de OR lógico

OR lógico pode ser usado para converter um caractere de maiúscula para minúscula, já que a diferença entre o código ASCII dos caracteres maiúsculos e minúsculos é constante, isto é, 32 (2^5).

Exemplo: Altere o caractere A (código ASCII 65) para a (código ASCII 97).

A	65	0000000001000001
OR 2^5	32	0000000000100000
a	97	0000000001100001

Este método é totalmente flexível, porque se você tentar converter um caractere maiúsculo em minúsculo, nada acontece. Tente você mesmo.

XOR

XOR, OR exclusivo retorna verdadeiro (1) quando X e Y são diferentes.

XOR	X	Y	X OR Y
	0	0	0
	0	1	1
	1	0	1
	1	1	0

Exemplo: 100 XOR 50

X	100	0000000001100100
XOR Y	50	0000000000110010
	86	0000000001010110

XOR pode ser calculado usando NOT, AND e OR. Eis como fazer.

X XOR Y = (X AND(NOT Y)) OR ((NOT X)AND Y)

Dessa forma, em português claro, XOR é ou X E NÃO Y OU NÃO X E Y.

X XOR Y = (X AND(NOT Y)))OR((NOT X)AND Y)

Prova:

X	Y	NOT X	NOT Y
0	0	1	1
1	0	0	1
0	1	1	0
1	1	0	0

X	(NOT Y)	X AND(NOT Y)
0	1	0
1	1	1
0	0	0
1	0	0

(NOT X)	Y	(NOT X) AND Y
1	0	0
0	0	0
1	1	1
0	1	0

(X AND(NOT Y))	((NOT X) AND Y)	(X AND (NOT Y)) OR ((NOT X) AND Y)
0	0	0
1	0	1
0	1	1
0	0	0

Assim, X XOR Y = (X AND(NOT Y)) OR ((NOT X) AND Y)

Exemplo: Uso de XOR

XOR tem a propriedade única tal que se você avaliar um valor XOR com outro valor duas vezes, você terá o valor original de volta.

X XOR Y XOR Y = X

Vamos tentar 100 XOR 50 XOR 50 como exemplo:

100 XOR	50	
X	100	0000000001100100
XOR Y	50	0000000000110010
100 Y	86	0000000001010110
(100 XOR	50) XOR 50	
X	86	0000000001010110
XOR Y	50	0000000000110010
	100	0000000001100100

Dessa forma 100 XOR 50 XOR 50 = 100

EQV

Esta é a função de equivalência lógica. EQV retorna verdadeiro sempre que X e Y forem iguais e falso quando diferentes.

EQV	X	Y	X EQV Y
	0	0	1
	1	0	0
	0	1	0
	1	1	1

Exemplo: 51 EQV 75

X	51	0000000000110011
EQV Y	74	0000000001001010
	–122	1111111110000110

A seguinte relação é verdadeira.

X EQV Y = (X AND Y) OR ((NOT X) AND (NOT Y))

Prova:

X	Y	NOT X	NOT Y
0	0	1	1
1	0	0	1
0	1	1	0
1	1	0	0

X	Y	(X AND Y)
0	0	0
1	0	0
0	1	0
1	1	1

(NOT X)	(NOT Y)	((NOT X) AND (NOT Y))
1	1	1
0	1	0
1	0	0
0	0	0

(X AND Y)	((NOT X) AND (NOT Y))	(X AND Y) OR ((NOT X) AND (NOT Y))
0	1	1
0	0	0
0	0	0
1	0	1

Dessa forma,

X EQ Y = (X AND Y)OR((NOT X) AND (NOT Y))

IMP

Tabela da verdade IMP

IMP	X	Y	X IMP Y
	0	0	1
	1	0	0
	0	1	1
	1	1	1

100 IMP 50 pode ser calculado como a seguir:

X	100	0000000001100100
IMP Y	50	0000000000110010
	−69	1111111110111011

A seguinte relação é verdadeira.

X IMP Y = (NOT X) OR Y

Prova:

X	Y	NOT X
0	0	1
1	0	0
0	1	1
1	1	0

(NOT X)	Y	(NOT X) OR Y
1	0	1
0	0	0
1	1	1
0	1	1

Dessa forma, X IMP Y = (NOT X) OR Y

Vimos o que o MSX faz com relação à álgebra de Boole. Existem muitas outras funções úteis como NOR e NAND que não estão incluídas no MSX mas que podem ser simuladas usando NOT, AND e OR. Lembre-se de que essas três funções são a base da álgebra booleana e que podem ser combinadas para formar outras.

Dê uma olhada na tabela a seguir: duas variáveis binárias, X e Y, que podem ter cada uma o valor 1 ou 0. Elas podem ter quatro combinações de 0 e 1, já que 2^2 = 4. Existem 16 combinações dessas combinações (ou funções), já que 2^2^2 = 16. O que você tem é uma lista de todas as combinações possíveis ou tabelas da verdade.

X	Y	f0	f1	f2	f3	f4	f5	f6	f7	f8	f9	f10	f11	f12	f13	f14	f15
0	0	0	0	1	0	1	0	0	0	1	0	1	1	1	0	1	1
0	1	0	0	1	1	0	0	0	1	0	1	1	0	1	1	0	1
1	0	0	1	0	0	1	0	1	0	0	1	0	1	1	1	0	1
1	1	0	1	0	1	0	1	0	0	0	1	1	1	0	0	1	1

Você pode reconhecer o seguinte:

```
f2   é  NOT
f5   é  AND
f9   é  OR
f13  é  XOR
f11  é  IMP
f14  é  EQV
```

Elas estão todas no MSX-BASIC; mas, e a respeito das outras, como as f1, f3 etc?

Nesta seção iremos ver os operadores lógicos que não são do MSX. Eles podem ser usados no MSX como expressões equivalentes feitas com NOT, AND e OR.

f0 Sempre função falsa, qualquer que seja a combinação de X e Y. Não usada em computação.

f1 Sempre X, qualquer que seja o valor de Y.

f2 NÃO X (veja NOT).

f3 Sempre Y, qualquer que seja o valor de X.

f4 NOT Y (veja NOT).

f5 X AND Y (veja AND).

f6 X AND (NOT Y).

f7 (NOT X) AND Y.

f8 Função NOR. Seus equivalentes no MSX são (NOT X) AND (NOT Y) ou, alternativamente, NOT (X OR Y) ... Veja a Lei de De Morgan.

f9 X OR Y (veja OR lógico).

f10 ((NOT X) OR Y) Similar a IMP.

f11 X IMP Y (veja IMP). Similar a f10. Alternativamente X OR (NOT Y).

f12 Função NAND. Ela é a negação de AND (NAND é a abreviatura de NOT AND).

NOT (X AND Y)
(NOT X) OR (NOT Y)
X NAND Y = (NOT X) OR (NOT Y) = NOT (X AND Y)
Veja a Lei de De Morgan mais adiante.

f13 X XOR Y. OR exclusivo (veja XOR). Alternativamente (X AND (NOT Y)) OR ((NOT X)) AND Y).

f14 X EQV Y. Equivalência lógica.
X EQV Y = (X AND Y) OR ((NOT X) AND (NOT Y)).

f15 Sempre verdadeiro, qualquer que seja o valor de X e Y. Não usado.

NAND e NOR, ambos não incluídos no MSX-BASIC, são funções úteis para se lembrar.

ALGUMAS RELAÇÕES LÓGICAS ÚTEIS

Existe uma variedade de relações lógicas que podem ser usadas na álgebra booleana, por exemplo, para reduzir expressões lógicas muito longas através da minimização.

USANDO OR LÓGICO

1. X OR 0=X
2. X OR X=X
3. X OR X OR X OR X OR...=X
4. X OR (NOT X) = 1
5. X OR 1 = 1
6. X OR (X AND Y) = X
7. X OR ((NOT X) AND Y) = X OR Y

USANDO AND LÓGICO

1. X AND 0=X
2. X AND X=X
3. X AND X AND X AND X AND...=X
4. X AND (NOT X) = 1
5. X AND 1 = 1
6. X AND (X AND Y) = X AND Y
7. X AND ((NOT X) AND Y) = 0

Exemplos:

Tente alguns dos seguintes exemplos em seu computador e veja se as relações abaixo são realmente verdadeiras:

	RESPOSTAS
PRINT 145 OR 0	145
PRINT 167 OR 167	167
PRINT 167 OR 167 OR 167	167
PRINT 133 OR (133 AND 222)	133
PRINT 111 AND 0	0
PRINT 111 AND 111	111
PRINT 111 AND 111 AND 111 AND 111	111
PRINT 234 AND (NOT 234)	0
PRINT 234 AND (234 OR 54)	234

LEIS DE DE MORGAN

À parte das relações anteriores, a álgebra booleana tem dois truques especiais para reduzir o comprimento das expressões lógicas.

1. **Negação do OR lógico**
   A negação (NOT) de um OR lógico é o equivalente ao AND lógico de negações das variáveis significando o OR lógico. Isso pode ser expresso por:

   NOT (X OR Y) = (NOT X) AND (NOT Y)

2. **Negação do AND lógico**
   A negação de um AND lógico é o equivalente de um OR lógico das negações das variáveis relacionadas ao AND lógico. Isso pode ser expresso por:

   NOT (X AND Y) = (NOT X) OR (NOT Y)

LEIS DE REARRANJAMENTO

Em matemática, existem leis que permitem que você rearranje e simplifique expressões complexas. Elas são conhecidas como as leis comutativa, associativa e distributiva. A álgebra booleana tem leis similares como as seguintes:

1. **Leis comutativas**

   a) X OR Y = Y OR X

   b) X AND Y = Y AND X

2. **Leis associativas**

   a) X AND (Y AND Z) = (X AND Y) AND Z = X AND Y AND Z

   b) X OR (Y OR Z) = (X OR Y) OR Z = X OR Y OR Z

3. Leis da distribuição

a) X AND (Y OR Z) = (X AND Y) OR (X AND Z)

b) X OR (Y AND Z) = (X OR Y) AND (X OR Z)

Resumo das Tabelas da Verdade dos Operadores Lógicos do MSX

X	Y	AND	OR	XOR	EQV	IMP
0	0	0	0	0	1	1
1	0	0	1	1	0	0
0	1	0	1	1	0	1
1	1	1	1	0	1	1

CAPÍTULO 35

ÁLGEBRA BOOLEANA II: O IF/THEN/ELSE

Quando você está escrevendo um programa de computador, é muito provável que o computador tenha de tomar decisões por si mesmo enquanto estiver rodando o seu programa. Para tomar decisões sob certas condições, use o comando IF para primeiro testar as condições, e então executar certas tarefas dependendo do resultado. IF é sempre usado junto com THEN e algumas vezes com ELSE. Existem dois tipos de formatos básicos:

IF <condições> THEN <comandos>

e

IF <condições> THEN <comandos> ELSE <comandos>

IF é seguido pela condição a ser testada. Se <condições> for verdadeira, então o computador executa os <comandos> após a palavra THEN. Se o computador descobrir que <condições> não é verdadeiro, se não existir a palavra ELSE nessa linha, ele irá executar a linha seguinte sem executar <comandos> após o THEN.

Entretanto, se existir uma palavra ELSE dentro dessa linha e <condições> for falsa, os <comandos> após ELSE serão executados. Neste caso <comandos>, entre THEN e ELSE são completamente ignorados.

Para iniciar, vamos dar uma olhada na parte <condições> da estrutura IF <condições> THEN...ELSE.

<CONDIÇÕES >

As <condições> seguem todas as regras e regulamentos descritos nos capítulos "Expressões e Operadores" e "Álgebra Booleana". Esta seção mostra o lado prático do que nós dissemos nos últimos dois capítulos.

COMPARAÇÃO NUMÉRICA

A condição mais simples compara dois valores; por exemplo, elas podem comparar se um número é maior que outro. Elas usam os operadores relacionais, como <, >, <>, =, <=, >=.

Exemplos:

```
1. 15 < 8        retorna 0 (falso)
2. a%=67         retorna  -1 (verdadeiro se a%=67)
                           0 (falso se a% não for 67)
```

No formato IF THEN:

```
1. IF D%=100 THEN PRINT "D% É 100"
2. IF V < =G THEN PRINT "V < =G"
```

Existem formas mais simples, mas você pode também usar expressões aritméticas em ambos os lados do operador relacional.

```
1. IF F%*H%=9876 THEN PRINT "VERDADEIRO"
2. IF 2^8=N%-1919*Y% THEN PRINT "VERDADEIRO"
```

Também não existe razão por que você não possa usar funções do BASIC como COS, INSTR etc. De fato, você pode mesmo incorporar uma função FN definida pelo usuário, também.

```
1. IF INSTR(A$,B$)=5 THEN PRINT B$
2. IF COS(0.6242) > =TAN(D%) THEN PRINT "VERDADEIRO"
3. IF FNA(D%*I%-100)=&HFF THEN PRINT "FF"
```

CONDIÇÃO NUMÉRICA SEM COMPARAÇÃO

Não é estritamente necessário ter uma relação como <condição>. De fato, ela pode ser uma variável numérica simples ou somente uma expressão. Dê uma olhada neste exemplo:

```
IF X% THEN PRINT "X%=VERDADEIRO"
```

Neste caso o computador toma a condição como verdadeira quando a variável X% for diferente de zero e falsa quando X% = 0.

Geralmente:

```
IF <expressão> THEN <comando>

<expressão> = Verdadeira, quando <expressão> retornar não zero.
<expressão> = Falsa, quando <expressão> retornar 0.
```

Você pode usar isso em uma grande variedade de situações. Eis alguns exemplos para ilustrar este ponto.

1. IF A < > 0 THEN PRINT "A NÃO 0"
   ... pode ser
   IF A THEN PRINT "A NÃO 0"
2. Para verificar se existe um certo caractere dentro de uma string.
   IF INSTR(S$,"A") THEN PRINT "S$ CONTÉM O CARACTERE 'a'"

COMPARAÇÃO DE STRINGS

Você pode comparar duas strings usando operadores relacionais de modo muito semelhante. A comparação é feita tomando um caractere por vez de cada string e comparando os códigos ASCII. Se os códigos ASCII diferirem, o número de código de minúscula precede o maior. Para aqueles caracteres que correspondam ao código ASCII, veja a tabela de códigos ASCII. Se durante a comparação de strings o final de uma string

for encontrado, a string mais curta será dita como sendo a menor. Comparação de strings podem ser usadas para colocá-las em ordem alfabética.

No seguinte exemplo, todas as condições são verdadeiras.

```
1. "abc" = "abc"
2. "ABC" < "abc"
3. "ABC" < "ABD"
4. "ab" < "abc"
5. "1234" < "12345"
```

CONDIÇÕES MÚLTIPLAS DE TESTE USANDO OPERADORES BOOLEANOS

É possível testar condições múltiplas em uma rodada, usando operadores lógicos booleanos. Existem 6 operadores lógicos no MSX-BASIC, mas somente 3 deles, NOT, AND e OR, são relevantes. Eles governam todas as regras da álgebra booleana, como as Leis de De Morgan.

1. Uso simples do AND.
IF <condição–1> OR <condição–2> THEN <comando>.
Neste caso, se somente ambas as condições forem verdadeiras, o computador irá executar o comando após o <comando>, isto é:

```
IF X=0 AND Y$="SIM" THEN PRINT "ÓTIMO"
```

2. Uso simples do OR.
IF < condição–1> OR <condição–2> THEN <comando>.
Neste caso, se uma ou ambas as condições for verdadeira, então computador irá executar o <comando> após THEN, por exemplo:

```
IF X=0 OR Y=0 THEN PRINT "UM DELES É ZERO"
```

3. Uso simples do NOT.
IF NOT <condição> THEN <comando>.

Se a condição for verdadeira, então NOT <condição> retorna o oposto, isto é, falso; e vice-versa, por exemplo:

```
IF NOT (V=U*8) THEN PRINT "V NÃO É U*8"
```

Agora, você deve ter notado que não tem sentido usar NOT na relação abaixo, quando você pode reescrevê-la como:

```
IF V < > U*8 THEN PRINT "V NÃO U*8""
```

Eis uma lista de tais relações que têm o mesmo significado.

```
NOT(X=Y)          X <  >
NOT(X <  > Y)     X=Y
NOT(X < =Y)       X < Y
NOT(X > =Y)       X < Y
NOT(X > Y)        X < =Y
NOT(X < Y)        X > =Y
```

Desse modo, não desperdice a memória do seu computador. O que NOT pode fazer é o reverso do resultado de uma expressão como:

```
IF NOT INSTR(A$,"y") THEN PRINT "Y NÃO ESTÁ NA STRING A$"
```

O exemplo a seguir é auto-explicativo.

MINIMIZAÇÃO – A LEI DE DE MORGAN

NEGAÇÃO DO OR LÓGICO:

Os três exemplos abaixo têm o mesmo efeito:

```
IF NOT (X=9 OR Y=10)
IF NOT (X=9) AND NOT (Y=10)
IF X <  > 9 OR Y <  > 10
```

Se estiver em dúvida, tente-os e veja por si próprio.

ESTRUTURAS IF/THEN/ELSE

ELSE é uma parte da estrutura IF/THEN/ELSE. ELSE diz ao computador que se a <condição> em IF não for satisfeita (isto é, falsa), então pule os comandos após THEN e execute os comandos após ELSE.

Comandos IF/THEN/ELSE múltiplos podem ser definidos bem como estruturas IF/THEN/ELSE. Elas só são limitadas pelo tamanho de uma linha, que contém 255 caracteres. Se existir menos cláusulas ELSE que THEN, então cada ELSE é comparado com o THEN mais próximo.

O segundo comando IF é executado após o primeiro comando ter sido dado como falso.

Sintaxe

O formato IF...THEN GOTO <linha> pode ser abreviado de duas formas:

1. IF...THEN <linha>
2. IF...GOTO <linha>

ELSE GOTO <linha> pode ser também abreviado:

1. IF...THEN...ELSE <linha>
2. IF...THEN...GOTO <linha>

Pontos a serem lembrados

Os comandos IF/THEN/ELSE tendem a se tornar muito longos para caber em uma única linha, porque você pode ter múltiplos comandos após THEN e ELSE. Se for o caso, então é aconselhável usar sub-rotinas usando o comando GOSUB.

É bom evitar o uso de comandos IF/THEN/ELSE em níveis, pois eles podem trazer muitos problemas. Você pode criar um programa tipo "espaguete" sem querer.

CAPÍTULO 36

IMPRIMINDO COM O USING

INTRODUÇÃO

O MSX tem um controle de formato de impressão bastante simples e poderoso. O comando PRINT USING permite que você imprima strings e números em vários formatos.

Sintaxe

PRINT USING <expressão–string> ; <lista de itens>

Existem duas partes no argumento de PRINT USING – a string de especificação do formato e a lista de itens a ser impressa. Elas são separadas por ponto e vírgula. A <lista de itens> pode ser número ou string. <expressão–string> contém a string especial de formatação que determina como os itens devem ser impressos. Ela pode ser ou uma string ou uma variável string.

EXPLICAÇÃO DE CARACTERES USADOS NO COMANDO PRINT USING

! O ponto de exclamação especifica que o primeiro caractere da string é para ser impresso.

Eis um programa-exemplo, que imprime somente as iniciais do primeiro nome e sobrenomes em linhas DATA.

```
10 FOR I=1 TO 3
20 READ NOME$,SBRNOME$
30 PRINT USING "!";NOME$;".",SBRNOME$
40 NEXT I
50 DATA JOSE,MARCIO
60 DATA MAX,RICK
70 DATA ANDRE,MAD
```

& Insere uma string especificada na posição dada por &.

```
10 A$="QWERTY!"
20 PRINT USING "ESTE E UM TECLADO &";A$
```

\# O sinal "#" indica um dígito a ser impresso. Por exemplo: PRINT USING "#.###";1.3 imprime

1.300

isto é, ele formata o número a ser exibido.

Como você pôde ver no exemplo anterior, você pode incluir um ponto decimal na string de formatação, entre os sinais "#". Se o número tiver menos dígitos que os especificados pela string de formatação, ele será alinhado à direita com espaços precedendo-o.

```
10 PRINT "####.#######"
20 PRINT USING "####.#######"; ATN(1)*4

####.#######
   3.1415927
```

Se os caracteres de formatação especificarem mais casas decimais que o número a ser exibido, os espaços vazios serão preenchidos com zeros.

```
10 PRINT "##.######"
20 PRINT USING "##.######";99.999

##.######
99.999000
```

Os números serão arredondados se não couberem no campo especificado.

```
10 PRINT "##.###"
20 PRINT USING "##.###";10.2367

##.###
10.237
```

Você pode imprimir dois ou mais números com o mesmo formato em um comando PRINT USING.

```
10 PRINT USING "##.##    ";9.866,18.7

 9.87    18.70
```

Se o número a ser impresso for negativo, então o sinal negativo será exibido, mas ele irá ocupar um dígito de espaço no especificador.

```
10 PRINT "###.####"
20 PRINT USING "###.####";-2.63

###.####
 -2.6300
```

Se o número a ser impresso não couber no campo especificado pelos caracteres de formatação, um sinal de percentagem (%) será impresso na frente do número. Isto acontece quando o número arredondado exceder ao tamanho do campo.

```
10 PRINT USING "#.###";9.9999

%10.000
```

\+ Um sinal de mais no início ou no final da string de formatação irá fazer com que o sinal do número seja impresso, isto é, um "+" ou "–" na posição especificada.

```
10 PRINT USING "+###.#####";-.123422,10.986

 -0.12342 +10.98600

10 PRINT USING "####.#+";10.71,100,-200.5

  10.7+ 100.0+ 200.5-
```

** O sinal de duplo asterisco faz com que os espaços de sobra no campo numérico sejam preenchidos com "*". Eles representam também mais dois dígitos de espaços no campo.

```
10 PRINT "**##.#########"
20 PRINT USING "**##.#########";ATN(1)*4
30 PRINT USING "**##.#########";-ATN(1)*4

**##.########
***3.14159265
**-3.14159265
```

$$ O sinal de duplo cifrão coloca uma sigla de cifrão na frente de um número. $$ especifica mais dois dígitos de espaço, um dos quais será o sinal de cifrão.

```
10 PRINT"$$+#.#########"
20 PRINT USING "$$+#.#########";ATN(1)*4
30 PRINT USING "$$+#.#########";-ATN(1)*4

$$+#.########
 $+3.141559265
 $-3.141559265
```

Observe que você não pode usar o formato exponencial junto com o sinal $$.

****$** Este sinal é uma combinação dos sinais ** e $S. Todos os espaços em branco são preenchidos por * e o número é precedido por um sinal $. **S especifica mais três posições de dígitos, uma das quais é o sinal $.

```
10 PRINT " **$#.########"
20 PRINT USING "**$+#.########";ATN(1)*4
30 PRINT USING "**$+#.########";-ATN(1)*4

 **$#.########
**$+3.14159265
**$-3.14159265
```

, Uma vírgula é colocada à esquerda do ponto decimal na string de formatação para fazer com que uma vírgula seja impressa a cada três dígitos à esquerda do ponto decimal.

```
10 PRINT"##########,.##"
20 PRINT USING"##########,.##";1090382.88#
30 PRINT"$$##########,.##"
40 PRINT USING"$$##########,.##";1090382.88#

##########,.##
  1,090,382.88
$$##########,.##
  $1,090,382.88
```

Uma vírgula no final de uma string de formatação imprime uma vírgula no final do número.

```
10 PRINT USING "##.##,";12.567

12.57,
```

^^^^ Este é o especificador de formatação do expoente. Ele gera os espaços para E+xx. Você também pode especificar a posição dos pontos decimais. Dígitos significativos são alinhados à esquerda e o expoente é ajustado.

```
PRINT USING"##.##^^^^";200.00

 2.00E+02

PRINT USING"+#.##^^^^";200.00

+2.00E+02

PRINT USING"#.##^^^^+";-200.00

2.00E+02-
```

Notas:

Se o número de dígitos especificados exceder a 24, irá provocar um erro "ILLEGAL FUNCTION CALL".

LPRINT USING

LPRINT USING é exatamente o mesmo que PRINT USING, porém ele imprime na impressora e não no modo de tela.

PRINT# USING

PRINT# USING é exatamente o mesmo que PRINT USING, porém ele imprime no dispositivo especificado pelo número após a marca #. Você deve executar um comando OPEN para abrir um canal para o dispositivo onde você deseja imprimir, antes de usar este comando.

Sintaxe

PRINT# <número de arquivo>, USING <expressão–string> ; <lista de expressões>.

CAPÍTULO 37

TRATAMENTO DE INTERVALOS E INTERRUPÇÕES PELO BASIC

INTRODUÇÃO

O computador MSX é equipado com vários comandos de tratamento de intervalos. Tratamento de intervalos (ou eventos) é dirigido por interrupções. Quer dizer que, enquanto o computador está executando um programa, ele também está examinando se um evento em particular acontece. Se esse evento ocorrer, ele interrompe os procedimentos imediatamente saltando para uma sub-rotina de usuário predeterminada.

O MSX pode lidar com interrupções para os seguintes eventos:

1.	Definir intervalo de tempo	ON INTERVAL GOSUB
2.	Pressionar teclas de função	ON KEY GOSUB
3.	Pressionar <CTRL> <STOP>	ON STOP GOSUB
4.	Pressionar gatilho	ON STRIG GOSUB
5.	Colisão de "sprites"	ON SPRITE GOSUB
6.	Erros	ON ERROR GOSUB

1. INTERRUPÇÃO DE INTERVALO DE TEMPO

Comandos usados:

ON INTERVAL = <tempo> GOSUB <linha>
INTERVAL ON/OFF/STOP

O MSX tem o seu próprio relógio interno, que começa a funcionar quando você liga o computador. O tempo pode ser visto com a função TIME. TIME é incrementado a cada 1/60 avos de segundo, a cada vez que o processador de exibição de vídeo fizer uma interrupção.

Usando ON INTERVAL GOSUB, você pode especificar um intervalo de tempo para a interrupção ocorrer. Como TIME é incrementado a cada 1/60 avos de segundo, para definir o intervalo de tempo para interromper a cada 1 segundo, você tem de comparar o INTERVALO com 60, isto é, ON INTERVAL=60 GOSUB <linha>.

ON INTERVAL = <tempo> GOSUB também define a sub-rotina para o qual o BASIC deve saltar sempre que a interrupção ocorrer. O computador irá saltar para essa sub-rotina assim que a interrupção ocorrer, não importa em que linha do programa esteja executando.

O intervalo de interrupção é ativado por INTERVAL ON. Isso diz ao computador para iniciar a contagem do relógio. A menos que seja executado este comando ON INTERVAL GOSUB não irá funcionar.

Uma vez ocorrida a interrupção de tempo, é executado automaticamente INTERVAL STOP. Isto evita que a interrupção de tempo ocorra durante a interrupção de tempo da atual sub-rotina. Entretanto, ele se lembra se uma interrupção ocorreu durante a atual sub-rotina; neste caso, o computador irá executar imediatamente a sub-rotina outra vez, a menos que a atual sub-rotina desative completamente o interruptor de tempo executando INTERVAL OFF.

Após deixar a sub-rotina de interrupção, o computador irá executar INTERVAL ON automaticamente para ativar a interrupção, a menos que INTERVAL OFF seja executado dentro dessa sub-rotina.

Você pode desativar o interruptor de tempo a qualquer momento que queira, usando INTERVAL OFF.

O interruptor de tempo é desativado assim que o BASIC sair de um programa, isto é, não é possível interrupção de tempo em modo direto. Ele é também desativado em qualquer sub-rotina de desvio de erro.

Exemplo:

Um programa curto para simular um exame digital com bips:

```
10 ON INTERVAL=60 GOSUB 60
20 INTERVAL ON
30 CLS
40 S=0:M=0
50 GOTO50
55 REM sub rotina de intervalo
60 IF S=60 THEN S=0:M=M+1:LOCATE 10,11:PRINT "min";M:B
EEP
70 S=S+1
80 LOCATE 10,10:PRINT "seg";S
90 BEEP
100 RETURN
```

LINHA 10	DEFINE O TEMPO PARA 1 SEGUNDO NA SUB-ROTINA EM 60
LINHA 20	ATIVA O DESVIO INTERNO
LINHA 30	LIMPA A TELA
LINHA 40	INICIALIZA S E M
LINHA 50	LOOPING INFINITO PARA ESPERAR UMA INTERRUPÇÃO
LINHA 60	SE 60 SEGUNDOS, ENTÃO 1 MINUTO
LINHA 70	S+1 SEGUNDOS
LINHA 80	IMPRIME SEGUNDOS
LINHA 90	TOCA O BIP
LINHA 100	RETORNA PARA ONDE ELE DEIXOU A ROTINA PRINCIPAL: NESTE CASO SEMPRE NA LINHA 50

```
SEG 54
MIN 3
```

2. INTERRUPÇÃO DE TECLA DE FUNÇÃO

Comandos usados:

ON KEY OSUB <linha>, <linha>, ...
KEY (<número de tecla>) ON/OFF/STOP

É possível definir interrupções para as 10 teclas de função usando ON KEY GOSUB e KEY ( ) ON/OFF/STOP.

ON KEY GOSUB é seguido por uma lista de números, especificando a sub-rotina de interrupção para cada tecla de função. Se não existir uma sub-rotina correspondente para uma tecla de função em particular, então o número da linha pode ser omitido.

O comando KEY (<número de tecla>) ON ativa interrupções de teclas de funções para cada tecla de função. A menos que esse comando seja executado, o computador não irá procurar por uma interrupção para a tecla de função. Quando houver uma interrupção, o computador salta para a sub-rotina especificada em ON KEY GOSUB.

Uma vez ocorrida a interrupção de tecla de função, é executada automaticamente KEY (<número de tecla>) STOP. Isto evita que outra interrupção pela mesma tecla de função ocorra durante a atual sub-rotina de interrupção. Entretanto, ele lembra se essa tecla de interrupção foi pressionada durante a atual sub-rotina e o computador irá executar automaticamente a mesma sub-rotina outra vez, a menos que a atual sub-rotina desative totalmente a TECLA de interrupção executando KEY (<número de tecla>) OFF.

```
10 ON KEY GOSUB 100,120,140,160,180,200
20 KEY(1) ON
30 KEY(2) ON
40 KEY(3) ON
50 KEY(4) ON
60 KEY(5) ON
70 KEY(6) ON
80 GOTO 80
90 REM sub-rotina
100 PRINT"funcao 1"
110 RETURN
120 PRINT"funcao 2"
130 RETURN
140 PRINT"funcao 3"
150 RETURN
160 PRINT"funcao 4"
170 RETURN
180 PRINT"funcao 5"
190 RETURN
200 PRINT"funcao 6"
210 RETURN
```

LINHA 10	DEFINE AS SUB-ROTINAS PARA F1, F2, F3, F4, F5 E F6
LINHA 20	ATIVA A TECLA DE FUNÇÃO F1
LINHA 30	ATIVA A TECLA DE FUNÇÃO F2

LINHA 40	ATIVA A TECLA DE FUNÇÃO F3
LINHA 50	ATIVA A TECLA DE FUNÇÃO F4
LINHA 60	ATIVA A TECLA DE FUNÇÃO F5
LINHA 70	ATIVA A TECLA DE FUNÇÃO F6
LINHA 80	LOOPING INFINITO PARA ESPERAR POR UMA TECLA DE FUNÇÃO
LINHA 100	ROTINA PARA F1
LINHA 110	RETORNA PARA O PONTO DE ONDE VEIO
LINHA 120	ROTINA PARA F2
LINHA 130	RETORNA PARA O PONTO DE ONDE VEIO
LINHA 140	ROTINA PARA F3
LINHA 150	RETORNA PARA O PONTO DE ONDE VEIO
LINHA 160	ROTINA PARA F4
LINHA 170	RETORNA PARA O PONTO DE ONDE VEIO
LINHA 180	ROTINA PARA F5
LINHA 190	RETORNA PARA O PONTO DE ONDE VEIO
LINHA 200	ROTINA PARA F6
LINHA 210	RETORNA PARA O PONTO DE ONDE VEIO

3. INTERRUPÇÃO <CTRL> <STOP> E DEFESA CONTRA BREAKS

Comandos usados:

```
ON STOP GOSUB <linha>
STOP ON/OFF/STOP
```

Um programa em BASIC é muito fácil de ser interrompido. Você só precisa pressionar <CTRL> <STOP>. Isto pára o programa imediatamente e coloca você de volta no modo direto. Entretanto, algumas vezes você vai achar que precisa de uma "defesa de interrupção" num programa para evitar que os usuários do programa vejam o seu conteúdo. Isto é feito desviando <CTRL> <STOP> com ON STOP GOSUB.

O comando ON STOP GOSUB define a sub-rotina para a interrupção <CTRL> <STOP>. Dentro dessa sub-rotina, você pode colocar ou uma mensagem dizendo "Não pode ser interrompido" ou somente um RETURN, de modo que o programa possa, imediatamente, executar a partir de onde houve a interrupção com <CTRL> <STOP>.

STOP ON/OFF/STOP ativa/desativa o desvio <CTRL> <STOP>.

Quando é executado STOP ON, o BASIC começa a verificar se <CTRL> <STOP> foi pressionada a cada vez que executa um novo comando. Se for detectada <CTRL> <STOP>, então o BASIC é desviado para a sub-rotina especificada no comando ON STOP GOSUB executado antes no programa.

Uma vez ocorrida uma interrupção, é executado um STOP STOP automaticamente. Isto pára a interrupção ocorrida durante a atual sub-rotina. Entretanto, ele lembra se <CTRL> <STOP> foi pressionada durante a atual sub-rotina e o computador irá imediatamente para a sub-rotina outra vez, se isso aconteceu, a menos que a atual sub-rotina desative completamente a interrupção <CTRL> <STOP>, executando STOP OFF. Após deixar a sub-rotina de interrupção, o computador irá executar automaticamente STOP ON para ativar a interrupção, a menos que seja executada STOP OFF dentro da sub-rotina.

A única maneira de interromper um programa a prova de interrupção é dar RESET no computador. De qualquer modo, você deve se lembrar de gravar o seu programa a prova de interrupção antes de executá-lo.

Exemplos:

Eis uma rotina pronta a "prova de interrupção" para incluir em seu programa.

```
10      ON STOP GOSUB 10000
20      STOP ON
..
..
   ESCREVA AQUI O SEU PROGRAMA
..
..
9999    REM Sub-rotina CTRL-STOP
10000   RTURN
```

LINHA 10	SUB-ROTINA STOP DEFINIDA PARA A LINHA 10000
LINHA 20	ATIVA O DETECTOR DE STOP
LINHA 10000	RETORNA PARA O LOCAL ONDE A INTERRUPÇÃO FOI DETECTADA

Observe que o desvio ON STOP não evita que a tecla STOP pare o programa. Ela somente evita que você interrompa o programa com <CTRL> <STOP>.

Se você pressionar somente a tecla STOP, você irá ver que ela pára o seu programa e exibe o cursor. Se você pressionar a tecla STOP pela segunda vez, o programa irá continuar.

A interrupção <CTRL> <STOP> é desativada quando o programa não sendo executado e também durante as rotinas de desvio de erros.

4. INTERRUPÇÃO DE DISPARO DE JOYSTICK (CONTROLADOR DE JOGOS)

Comandos usados:

```
ON STRIG GOSUB <uma lista de número de linhas>
STRIG (<n>) ON/OFF/STOP
```

O número de disparo, <n>, para cada disparo é conforme o mostrado:

0 = barra de espaços.
1 = disparo 1 para o *joystick* 1.
2 = disparo 2 para o *joystick* 2.
3 = disparo 1 para o *joystick* 3.
4 = disparo 2 para o *joystick* 4.

Cada *joystick* compatível com o MSX tem dois botões de disparo. A barra de <espaço> também é considerada como um gatilho.

ON STRIG GOSUB define as sub-rotinas de interrupção de disparo para esses gatilhos. Você pode definir sub-rotinas para todos os gatilhos de uma única vez listando os números de linhas para cada gatilho.

Se não existir sub-rotina para esse gatilho em particular, então você pode omiti-lo e colocar somente uma vírgula.

Por exemplo:

```
ON STRIG GOSUB 1000, 200,,,
```

irá interromper para uma sub-rotina quando:

a) gatilho 0 da barra de espaço

ou

b) gatilho 1 no joystick 1

for pressionado, mas nada acontece se qualquer um dos outros gatilhos for pressionado.

STRIG (<n>) ON ativa o desvio de gatilho para o gatilho especificado. Você somente pode especificar um gatilho por vez.

Uma vez ocorrida a interrupção, é executado um STRIG (<n>) STOP automático. Isto pára a interrupção que ocorreu durante a atual sub-rotina de gatilho. Entretanto, ele lembra se um disparador foi pressionado durante esta sub-rotina; e o computador vai imediatamente para a sub-rotina correspondente, para esse gatilho uma vez que tenha deixado a atual sub-rotina, a menos que a atual sub-rotina desative a interrupção para esse gatilho, executando STRIG (<n>) OFF. Após ter deixado a sub-rotina de gatilho, o computador irá automaticamente executar STRIG (<n>) ON para ativar a interrupção.

A interrupção STRIG é desativada quando o programa não estiver sendo executado e também durante as rotinas de desvio de erros.

Exemplos:

Eis um programa bem pequeno que demonstra como o desvio de gatilho funciona testando cada gatilho de *joystick*. O programa está escrito de modo que ambos os *joysticks* podem ser testados, bem como a barra de espaço. Se você pressionar a tecla <s>, o programa irá terminar.

```
10 ON STRIG GOSUB 100,120,140,160,180
20 STRIG(0) ON
30 STRIG(1) ON
40 STRIG(2) ON
50 STRIG(3) ON
60 STRIG(4) ON
70 IF INKEY$="s" THEN END
80 GOTO 70
```

```
90 REM ROTINA DOS DISPARADORES
100 PRINT "barra de espaco pressionada"
110 RETURN
120 PRINT "disparador 1 joystick 1"
130 RETURN
140 PRINT "disparador 1 joystick 2"
150 RETURN
160 PRINT "disparador 2 joystick 1"
170 RETURN
180 PRINT "disparador 2 joystick 2"
190 RETURN
```

LINHA 10	DEFINE TODAS AS SUB-ROTINAS DE DISPARO
LINHA 20	ATIVA O GATILHO 0
LINHA 30	ATIVA O GATILHO 1
LINHA 40	ATIVA O GATILHO 2
LINHA 50	ATIVA O GATILHO 3
LINHA 60	ATIVA O GATILHO 4
LINHA 70	SE <s> FOR PRESSIONADO TERMINA O PROGRAMA
LINHA 80	LOOPING DE RETORNO À LINHA 70
LINHA 90	ROTINAS DE DISPARO

5. INTERRUPÇÃO DE COLISÃO DE "SPRITES"

Veja o Capítulo de Gráficos *Sprites* Avançados para explicações detalhadas.

6. DETECÇÃO DE ERROS

Veja o Capítulo "Tratamento de Erros" para maiores detalhes.

CAPÍTULO 38

TRATAMENTO DE ERROS

MENSAGENS DE ERRO*

Quando o MSX-BASIC encontra um erro durante a execução de um programa ou comando, ele exibe uma mensagem de erro e pára a execução. Eis um resumo das mensagens de erro:

Código	Mensagem	Descrição
1	NEXT without FOR (ou NEXT sem FOR, no HOT-BIT)	
		Uma variável em um comando NEXT não coincide com uma variável em um comando FOR anterior; ou um comando NEXT foi encontrado sem um comando FOR anterior.
2	Sintax error (ou Erro de sintaxe, no HOT-BIT)	
		Erro de grafia no comando ou pontuação errada.
3	Return without GOSUB (ou "RETURN" sem "GOSUB" no HOT-BIT)	
		Um comando RETURN foi encontrado sem um comando GOSUB coincidente.
4	Out of DATA (ou sem "DATA", no HOT-BIT)	
		Um comando READ foi executado quando todos os dados em linhas DATA já foram usados; ou não existem comandos DATA para o comando READ ler.

* A princípio as mensagens de erros referem-se ao EXPERT. As mensagens do HOT-BIT vêm entre parênteses.

5 Illegal function call (ou Função ilegal, no HOT-BIT)

Um parâmetro ilegal foi passado para uma função matemática ou de string.

6 Overflow

O número é muito grande para o tipo de variável, função ou comando a ser tratado.

7 Out of memory (ou Falta memória, no HOT-BIT)

Saída da memória devido a um programa muito longo, com excesso de FOR, excesso de GOSUB, muitas funções ou variáveis.

8 Undefined line number (ou nº linha inexistente, no HOT-BIT)

Foi referido um número de linha não existente em um comando GOTO, GOSUB, IF/THEN/ELSE ou em um comando DELETE.

9 Subscript out of range (ou Índice fora do limite, no HOT-BIT)

Um subscrito de um elemento de uma matriz está fora da faixa especificada no comando DIM; ou o número do subscrito está errado.

10 Redimensioned array (ou "DIM" redefinido, no HOT-BIT)

Dimensionando uma matriz já existente usando DIM.

11 Division by zero (ou "DIVISÃO" por zero, no HOT-BIT)

Foi encontrada uma divisão por zero em uma expressão.

12 Illegal direct (ou "Direto ilegal", no HOT-BIT)

Um comando que não é permitido no modo direto.

13 Type mysmatch (ou "Tipo desigual", no HOT-BIT)

Uma variável string foi atribuída a um número ou vice-versa.

14 Out of string space (ou "Falta área 'string' ", no HOT-BIT)

Saiu fora da área de trabalho de strings na RAM do sistema.

15 String too long (ou " 'String' longa", no HOT-BIT)

Foi tentada a criação de uma string com mais de 255 caracteres.

16 String formula too complex (ou "String" complexa, no HOT-BIT)

Uma expressão de string é muito complexa para o computador tratar.

17 Can't continue (ou "Não contínuo!", no HOT-BIT)

Foi tentado continuar um programa que terminou com um erro, ou que já tenha sido editado ou terminado.

18 Undefined user function (ou "Função não definida", no HOT-BIT)

Foi chamada uma função FN sem primeiro ter sido definida com um comando DEF FN.

19 Device I/O error (ou, "Erro/Periférico", no HOT-BIT)

Um erro de entrada/saída em um cassete, impressora etc. É um erro fatal sem possibilidade de recuperação.

20 Verify error (ou "Erro/Verif.", no HOT-BIT)

Ao se usar o comando CLOAD?, a verificação de uma fita cassete falhou.

21 No RESUME (ou "RESUME", no HOT-BIT)

Uma rotina de desvio de erro não tem o comando RESUME que é esperado pelo computador.

22 RESUME without error (ou " 'RESUME' sem 'ERROR' ", no HOT-BIT)

Um comando RESUME não esperado foi encontrado fora de uma rotina de desvio de erro.

23 Unprintable error (ou "Erro indefinido", no HOT-BIT)

O erro foi encontrado mas não há mensagem.

24 Missing operand (ou "Falta operando", no HOT-BIT)

Uma expressão continha um operador sem operando seguindo-o.

25 Line buffer overflow (ou "Linha Muito Longa", no HOT-BIT)

Uma linha digitada tem caracteres sem excesso.

26-49 Unprintable error (ou "Erros indefinidos", no HOT-BIT)

Reservados para futura expansão.

50 Field overflow (ou "Campo maior", no HOT-BIT)

Um comando FIELD tentou alocar mais bytes que o especificado para o tamanho do registro de um arquivo aleatório em um comando OPEN; ou o final do buffer FIELD foi encontrado enquanto estava fazendo operação seqüencial de I/O (PRINT#, INPUT#) para um arquivo aleatório.

51 Internal error (ou "Erro interno", no HOT-BIT)

Mau funcionamento da máquina.

52 Bad file number (ou "Número do arquivo", no HOT-BIT)

Um comando se referiu a um arquivo não existente ou fora da faixa MAXFILES.

53 File not found (ou "Arquivo não existe", no HOT-BIT)

Um comando LOAD ou OPEN se referiu a um arquivo não existente.

54	File already open (ou "Arquivo aberto", no HOT-BIT)	
		Foi tentado abrir um arquivo já aberto.
55	Input past end (ou "Fim do arquivo", no HOT-BIT)	
		Um comando INPUT foi executado após todos os dados do arquivo já terem sido lidos. Use EOF para evitar que isso aconteça.
56	Bad file name (ou "Nome arquivo", no HOT-BIT)	
		Foi dado um nome de arquivo ilegal num comando LOAD, SAVE ou outro comando de I/O.
57	Direct statement in file (ou "Comando direto/Arquivo", no HOT-BIT)	
		Um comando foi encontrado durante a carga de um programa. Isso ocorre quando usamos CLOAD para carregar um programa que foi gravado no formato binário.
58	Sequential I/O only (ou "Arquivo seqüencial", no HOT-BIT)	
		Foi emitido um comando de acesso aleatório para um arquivo seqüencial.
59	File not OPEN (ou "Falta 'OPEN' ", no HOT-BIT)	
		O arquivo referido ainda não está aberto.
60-255	Unprintable error (ou "Erros indefinidos", no HOT-BIT)	
		Sem códigos de erros. Estes podem ser definidos pelo usuário.

TRATAMENTO DE ERROS

O MSX-BASIC contém muitas funções e comandos para ajudar a depurar o seu programa. É prática normal incorporar uma rotina de tratamento de erros ou de desvio dentro do seu próprio programa enquanto ele ainda está sendo desenvolvido. Isso irá ajudá-lo a detectar qualquer erro que você tiver cometido durante uma execução de teste do seu programa.

Antes de aprendermos como escrever tais rotinas de desvio de erros, vamos dar uma olhada em cada uma das palavras chaves do BASIC usadas no tratamento de erros.

ERL, nos dá a linha com erro.

ERL é uma variável reservada que contém o número da linha que contém o erro encontrado na execução do programa.

ERR indica o número do código do erro.

ERR contém o número de código de erro após o computador ter detectado um erro no programa. Ele tem um valor entre 1 e 255. Os significados dos erros associados ao número contido em ERR são listados da mesma forma que anteriormente.

ERROR

Existem duas maneiras básicas de se usar o comando ERROR.

1. Para simular a ocorrência de um erro. Um comando ERROR com um argumento inteiro levará o computador a pensar que tem um erro e ele terminará o programa e imprimirá a mensagem de erro com o número da linha.
2. Para criar erros definidos pelo usuário. Um comando ERROR junto com o uso de uma função de desvio de erro ON ERROR GOTO irá permitir que você crie seus próprios erros customizados (adiante daremos mais informações)

ON ERROR GOTO <linha>

ON ERROR GOTO permite desvio de erros e especifica para que linha o computador deve ir quando é detectado o erro. Uma vez dito ao computador para desviar nos erros, ele irá saltar para a linha especificada, para executar a sub-rotina de tratamento de erros especificada, se acontecer um erro durante a execução do programa ou em modo direto, isto é, fora do programa.

RESUME

RESUME significa reiniciar uma operação no BASIC após um procedimento de tratamento de erros ter sido executado usando o comando de desvio de erro ON ERROR GOTO. Após o erro ter sido tratado, RESUME diz ao computador para continuar a execução de acordo com a sintaxe abaixo:

RESUME ou RESUME 0

Reinicia a execução a partir do comando que provocou o erro.

RESUME NEXT

Reinicia a execução a partir do comando seguinte àquele que causou o erro.

RESUME <linha>

Reinicia a execução a partir do número de linha especificado.

ROTINAS DE TRATAMENTO DE ERROS

Para incorporar uma rotina de tratamento de erros em seu programa você deve fazer o seguinte:

1. Colocar o comando ON ERROR GOSUB no início do programa para permitir o desvio de erros daí para frente. Após esse comando ter sido executado, o computador irá, quando houver um erro, desviar para a rotina de tratamento de erros quer você esteja no modo direto ou indireto.
2. Colocar uma sub-rotina de tratamento de erros no final do programa.

Um programa simples com uma rotina de tratamento de erros se parece com este:

```
10 ON ERROR GOTO 100
20 PRINT 10
30 PRONT 20
40 PRINT 30
50 END
99 REM rotina de tratamento de erros
100 ON ERROR GOTO 0
```

LINHA 10	ATIVA O DESVIO DE ERROS E DEFINE A SUB-ROTINA DE ERRO PARA A LINHA 100
LINHA 30	ERRO AQUI
LINHA 50	TERMINA O PROGRAMA
LINHA 99	ON ERROR GOTO FAZ COM QUE O BASIC PARE E IMPRIMA O ERRO QUE CAUSOU O DESVIO

Quando o programa acima for executado, aparecerá o seguinte:

```
RUN
 10
Syntax error in 30
```

O erro da linha 30 foi detectado e o programa desviado para a linha 100. ON ERROR GOTO 0 tem o efeito de exibir o erro que causou o desvio e parar o programa. Ela também desativa o desvio de erro.

Até agora, tudo isso pode ser feito sem se precisar ter o desvio de erro. Se não existir um comando ON ERROR GOSUB, o computador ainda irá detectar o erro da linha 30 e dar a mensagem "Sintax error in 30" (erro de sintaxe em 30). Entretanto, a rotina de desvio de erro tem outros dispositivos.

Por exemplo, a sub-rotina de tratamento de erros pode conter um procedimento de recuperação de erro, para colocar você de volta no programa. Para fazer isso, você precisa saber que tipo de erros quer recuperar para tratá-los, e como o computador deve reiniciar a operação.

Neste exemplo, existem uns poucos itens de dados em comandos DATA na linha 60. Isto provoca um *erro de dados* (código 4) quando o computador estiver no seu quarto loop. A rotina de tratamento de erros é equipada com um procedimento de recuperação de erro na forma da linha 100. Essa linha descobre se o erro foi realmente um erro de dados verificando o código de erro de ERR, e então executa um comando RESTORE de modo que os dados possam ser usados outra vez. Ela reinicia a operação a partir da linha onde o erro foi detectado.

```
10 ON ERROR GOTO 100
20 FOR I=1 TO 5
30 READ A$
40 PRINT A$
50 NEXT I
60 DATA um,dois,tres
70 END
90 REM ROTINA DE ERROS
100 IF ERR=4 THEN RESTORE:PRINT "sem dados":RESUME
110 ON ERROR GOTO 0
```

LINHA 10	PERMITE O DESVIO DE ERROS E DEFINE A SUB-ROTINA DE ERRO PARA A LINHA 100
LINHA 20	EXECUTA O LOOP CINCO VEZES
LINHA 30	LÊ OS DADOS
LINHA 40	IMPRIME A$
LINHA 50	LOOP SEGUINTE
LINHA 60	SOMENTE TRÊS ITENS DE DADOS AQUI
LINHA 70	FIM
LINHA 100	SE ERR ESTIVER FORA DOS DADOS (CÓDIGO 4) ENTÃO RESTAURE E REINICIE A OPERAÇÃO A PARTIR DE ONDE ENCONTROU O ERRO
LINHA 110	IMPRIMA A MENSAGEM DE ERRO SE QUALQUER OUTRO ERRO FOR ENCONTRADO

```
run
um
dois
tres
sem dados
um
dois
```

Com alguns erros você não vai querer que o computador reinicie a operação a partir da mesma linha. Erros como erro de sintaxe não podem realmente ser remediados dentro de um programa, tendo de ser editado. Entretanto, se você quiser que o computador salte a linha com o erro, forneça a você uma mensagem, e prossiga então com o programa até o final, use o comando RESUME NEXT que reinicia a operação a partir da linha seguinte àquela em que o erro foi detectado.

```
10 ON ERROR GOTO 100
20 PRINT 10
30 PRONT 20
40 PRINT 30
50 END
99 REM ROTINA DE TRATAMENTO DE ERROS
100 PRINT"Erro de codigo=";ERR;"na linha ";ERL
110 RESUME NEXT
```

LINHA 10	ATIVA O DESVIO DE ERROS E DEFINE A SUB-ROTINA DE ERRO PARA A LINHA 100
LINHA 30	ERRO AQUI
LINHA 50	TERMINA O PROGRAMA
LINHA 100	EXIBE O CÓDIGO DE ERRO E A LINHA DE ERRO
LINHA 110	ENTÃO REINICIA A OPERAÇÃO A PARTIR DA PRÓXIMA LINHA

```
RUN
 10
ERRO DE CÓDIGO = 2 NA LINHA 30
 30
```

Observe que você pode especificar para que linha ir pelo comando RESUME, de modo que, no exemplo anterior, RESUME 40 tem o mesmo efeito.

É possível exibir a linha que contém o erro usando uma rotina de tratamento de erros. Será muito mais fácil corrigir o erro se a linha que contém o erro for exibida exatamente após o erro ter sido detectado. Para fazer isso você deve terminar a rotina de recuperação de erros com o comando LIST. (list ponto). LIST. tem o efeito de listar a última linha referida pelo computador.

```
10 ON ERROR GOTO 100
20 PRINT 10
30 PRONT 20
40 PRINT 30
50 END
99 REM ROTINA DE TRATAMENTO DE ERROS
100 PRINT"Erro de codigo=";ERR;"na linha ";ERL
110 LIST.
```

LINHA 10	ATIVA O DESVIO DE ERRO E DEFINE A SUB-ROTINA DE ERROS PARA A LINHA 100
LINHA 30	ERRO AQUI
LINHA 50	TERMINA O PROGRAMA
LINHA 100	EXIBE O CÓDIGO DO ERRO E A LINHA DO ERRO
LINHA 110	E ENTÃO LISTA A LINHA COM ERRO

```
10
ERRO DE CÓDIGO =2 NA LINHA 30
30 PRONT 20
```

Agora você pode corrigir a linha 30 para:

```
30  PRINT 20
```

COMO CRIAR OS SEUS PRÓPRIOS ERROS (ERROS CUSTOMIZADOS)

Um comando ERROR junto com o uso do comando de desvio de erro ON ERROR GOTO irá permitir que você crie os seus próprios erros customizados.

Existem cerca de 36 erros, entre os códigos de erro 1 e 60, na presente versão do MSX-BASIC. Números de códigos entre 61 e 255 podem ser usados pelo programador.

Para programar o seu próprio erro, primeiro você deve colocar o computador no modo de desvio de erro executando o comando ON ERROR GOTO <linha> no início do programa. Então, se, no meio do programa, uma condição aparecer de modo que você queira chamar a rotina de desvio de erro, você pode fazer isso com:

```
IF <condição> THEN ERROR <código de erro>
```

ON ERROR GOTO será ativado e o computador irá iniciar imediatamente a sub-rotina de desvio de erro. Dentro da sub-rotina você deve ter uma linha que diga:

```
IF ERR=<código de erro> THEN PRINT "Mensagem de erro"
```

No seguinte exemplo todos os comandos digitados com MATE serão tratados como um novo erro (nº 255), e são tratados na rotina de desvio de erro usando a variável ERR.

```
10 ON ERROR GOTO 100
20 INPUT"MEU MESTRE,QUAL O SEU DESEJO";A$
30 IF INSTR(A$,"MATE") THEN ERROR 255
40 PRINT"OK"
50 END
100 IF ERR=255 THEN PRINT "VOCE NAO PODE MATAR NADA NE
STA AVENTURA":RESUME 20
110 END
```

LINHA 10	ATIVA O DESVIO DE ERRO
LINHA 20	ENTRADA
LINHA 30	VERIFICA PALAVRA INVÁLIDA
LINHA 100	MENSAGEM DE ERRO REINICIA NA LINHA 20
LINHA 110	TERMINA SE O ERRO NÃO FOR 255

```
RUN
MEU MESTRE,QUAL O SEU DESEJO? MATE-O
VOCE NAO PODE MATAR NADA NESTA AVENTURA
MEU MESTRE,QUAL O SEU DESEJO? VAIPARA O LESTE
OK
```

É prática normal no MSX-BASIC, quando você está definindo os seus próprios códigos de erros, começar a partir do 255 para baixo, de modo que você possa evitar conflitos com qualquer futura expansão nas mensagens de erro feitas pelos fabricantes de computadores MSX.

NOTAS SOBRE O TRATAMENTO DE ERROS

O comando ERROR pode também ser usado para simular a ocorrência de um erro. Por exemplo:

```
ERROR 2
```

resultará em:

```
Sintax error
```

Se o código de erro não tiver mensagem de erro predefinida, então quando o computador encontra esse erro no programa, ele irá imprimir "Unprintable Error".

Nenhum desvio de erro é executado durante uma rotina de desvio de erros. Se existir um erro dentro da rotina de desvio de erros, ele irá dar uma mensagem de erro e parar o programa.

Nem todos os erros podem ser tratados por rotinas de desvio de erros. Dessa forma é recomendado terminar uma rotina de desvio de erros com ON ERROR GOTO 0 que irá parar a execução do programa BASIC e exibir a mensagem de erro.

ON ERROR desativa todos os desvios de tratamento de eventos como ON INTERVAL e ON STRIG.

CAPÍTULO 39

GRAVANDO NO E CARREGANDO DO CASSETE

INTRODUÇÃO

Existem diversos modos de se gravar e carregar programas e dados em um cassete. Este capítulo trata dos usos mais sofisticados do cassete.

COMANDOS E FUNÇÕES ASSOCIADAS COM A GRAVAÇÃO E CARGA DO CASSETE

Eis uma lista dos comandos e funções que são associados à exibição no modo de tela.

Gravar e carregar programas BASIC (veja a introdução ao MSX-BASIC para detalhes):

CLOAD	Carrega um programa BASIC do cassete.
CSAVE	Grava um programa BASIC do cassete.
CLOAD?	Verifica um programa BASIC em fita contra um na memória do computador.
MOTOR	Ativa/desativa o motor do cassete.

* Taxa da velocidade de gravação dos dados na fita k-7.

Gravar, carregar e "juntar" usando o formato ASCII.

SAVE — Grava um programa BASIC para um dispositivo especificado, em um arquivo ASCII.

LOAD — Carrega um programa BASIC gravado em um arquivo ASCII do dispositivo especificado.

MERGE — Junta (ou cola) um programa BASIC gravado em um arquivo ASCII com um programa BASIC na memória do computador.

Gravar e carregar uma seção na memória do computador como um programa em linguagem de máquina ou dados:

BSAVE — Grava uma seção de memória.

BLOAD — Carrega uma seção de memória.

Definir a taxa baud* para gravar no cassete:

SCREEN
CSAVE

A TAXA BAUD PARA GRAVAR NO CASSETE

A taxa baud é a taxa de velocidade na qual os dados são transmitidos para o cassete. O MSX usa o formato FSK que fornece a você a opção de duas taxas baud: 1.200 baud e 2.400 baud. Isso pode ser definido ou pelo comando SCREEN, ou pelo comando CSAVE. A taxa baud padrão é 1.200. Suas sintaxes são as seguintes:

```
SCREEN,,,1              1200 baud
SCREEN,,,2              2400 baud

CSAVE "<nome>"          1200 baud
CSAVE "<nome>",1        1200 baud
CSAVE "<nome>",2        2400 baud
```

A taxa baud de gravação pode ser ou 1.200 baud ou 2.400 baud, mas CLOAD, LOAD e BLOAD verificarão automaticamente em que taxa está gravado e carregar de acordo.

GRAVANDO E CARREGANDO NO FORMATO ASCII

Quando um programa normal em BASIC é gravado em uma fita cassete, isto ocorre em forma de símbolos, para que o programa possa ser gravado em um formato eficiente. Entretanto, se você precisar gravar o seu programa em código ASCII, deve usar o comando SAVE. Para carregar um programa gravado em código ASCII você usa o comando LOAD. Tanto para o SAVE como para o LOAD, você deve especificar o dispositivo no qual você está gravando ou do qual você está carregando na forma de um <descritor de dispositivo>. Entretanto, somente o cassete é suportado na atual versão do MSX-BASIC, de modo que o descritor de dispositivo deve sempre ser "CAS:". Aqui está uma lista da sintaxe:

SAVE "<descritor de dispositivo> <nome do programa>"
: Grava um programa BASIC em um arquivo ASCII.

LOAD "<nome do dispositivo>"
: Carrega o primeiro arquivo BASIC encontrado, gravado no formato ASCII.

LOAD "<nome do dispositivo> <nome do arquivo>"
: Carrega um arquivo BASIC com o nome especificado.

LOAD "<nome do dispositivo> <nome do arquivo>",R
: Carrega um arquivo BASIC com o nome especificado e então o executa.

Observe que a opção R no LOAD irá executar automaticamente o programa BASIC recém-carregado.

O uso principal da gravação no código ASCII é o de o programa poder ser intercalado com outro, que é o que irá acontecer com o próximo.

INTERCALANDO DOIS PROGRAMAS EM BASIC

Vamos dizer que você tem dois programas em BASIC, programa 1 e programa 2, e você quer intercalar o programa 2 com o programa 1, de forma que o programa 2 venha logo após o programa 1 quando ele for intercalado. Vamos dizer que ambos foram inicialmente gravados em fita cassete.

PROG1

```
10    REM PROGRAMA 1
20    PRINT "BEM VINDO"
30    PRINT "AO"
40    PRINT "MSX"
```

PROG2

```
10 REM PROGRAMA 2
20 FOR I=32 TO 58
30 PRINT CHR$(I)
40 NEXT I
```

Primeiro, carregue o programa 2 e renumere-o de modo que todos os números de linha sejam maiores que os do programa 1.

```
CLOAD "PROG 2"
RENUM 50
LIST
```

```
50 REM PROGRAMA 2
60 FOR I=32 TO 58
70 PRINT CHR$(I)
80 NEXT I
```

Então grave-o em um arquivo ASCII usando SAVE:

```
SAVE "CAS:PROG2"
```

Carregue o programa 1 do cassete:

```
CLOAD "PROG1"
```

Após você ter carregado o programa 1 do cassete, rebobine a fita até o ponto onde PROG2 está gravado. Digite:

```
MERGE "CAS:PROG2"
```

e ligue o cassete. O computador irá incluir o programa 2 no final do programa 1.

```
10 REM PROGRAMA 1
20 PRINT "BEM-VINDO"
30 PRINT "AO"
40 PRINT "MSX"
50 REM PROGRAMA 2
60 FOR I=32 TO 58
70 PRINT CHR$(I)
80 NEXTI
```

Favor observar o seguinte quando estiver intercalando dois programas:

1. Se um número de linha ocorrer tanto no programa inicial como no programa 2, o programa resultante irá conter a linha do programa 2.
2. MERGE precisa do nome de dispositivo em sua sintaxe.
   MERGE "<nome do dispositivo> <nome do arquivo>" <nome do dispositivo>=CAS: para cassete.
3. Se o nome do arquivo for omitido em um comando MERGE, então o primeiro arquivo ASCII encontrado na fita será intercalado.

GRAVANDO E CARREGANDO UMA SEÇÃO DA MEMÓRIA DO COMPUTADOR

Para gravar uma seção da memória do computador, de um endereço especificado de memória para o cassete, use o comando BSAVE. Você deve especificar o endereço inicial e final da memória que você quer gravar. Este comando é usado para gravar programas em linguagem de máquina e dados na forma de bytes.

Se você está gravando um programa em linguagem de máquina, você tem a opção de especificar o endereço de execução. Quando você está carregando o programa em linguagem de máquina com BLOAD, se o BLOAD especificar para auto-executar o programa em código de máquina, ele irá executar o programa no endereço especificado em BSAVE.

Qualquer coisa gravada com BSAVE deve ser carregada com BLOAD.

Se o endereço de execução for omitido quando o código for gravado, então o computador assume que o endereço de execução é o endereço inicial quando o programa

for carregado com a opção R de auto-execução do programa. Eis uma lista da sintaxe:

```
BSAVE "<nome do dispositivo> : <nome>", <endereço inicial>, <endereço final>
BSAVE "<nome do dispositivo> : <nome>", <endereço inicial>, <endereço final>,
      <endereço de execução>
BLOAD "<nome do dispositivo> : <nome>",R
```

<nome> e R são opcionais.

A opção R executa automaticamente o programa em código de máquina recém-carregado. R significa RUN (executar).

```
BLOAD "<nome do dispositivo> : <nome>", <número constante>
```

Quando o deslocamento (off-set) for especificado o arquivo será inserido na memória na posição especificada por <número constante>.

Os endereços podem ser dados em números hexadecimais, ou seja:

```
BSAVE "CAS:PROG",&HF300,&HF380,&HF30A.
```

CAPÍTULO 40

GRÁFICOS AVANÇADOS I

CARACTERÍSTICAS DE CADA MODO DE TELA

O MSX tem um dos mais versáteis chips gráficos, o TMS 9929A, desenvolvido pela Texas Instruments, Inc. dos Estados Unidos. É um dos mais compreensivos chips gráficos disponíveis, e fornece a você uma grande variedade de facilidades, como gráficos de alta resolução em 16 cores e animação de sprites. Ele pode até gerenciar sua própria RAM de 16K, que significa que o processador central não precisa fornecer memória para o gráfico no modo de tela.

O MSX-BASIC foi escrito especialmente com essas fortes capacidades gráficas e facilidades de programação para iniciantes. É fácil de ser usado uma vez que você já o tenha experimentado, e ele produzir algumas figuras espetaculares se você tentar com afinco.

Vamos iniciar explicando em primeiro lugar os modos de exibição disponíveis no MSX.

Modo de tela

Modo	Texto/Gráfico	Resolução	Cores	Entrada	Gráficos	Sprites
0	40x24 modo texto	40x24 car	2 em 16	sim	não	não
1	32x24 modo	32x24 car	2 em 16	sim	não	sim
2	gráf. alta resolução	256x192 carac.	16	não	sim	sim
3	Multicores	64x68 blocos	16	não	sim	sim

Notas:

ENTRADA: uso ou não do comando INPUT no modo de tela dado. Você não pode fazer entrada enquanto estiver em um modo gráfico.

GRÁFICOS: uso de comandos gráficos, por exemplo DRAW, no modo de tela dado. Geralmente você não pode usar comandos gráficos em modos de texto (0 e 1).

Comandos e funções associados a modos de tela

Eis uma lista de comandos e funções associados à exibição no modo de tela. Mais detalhes são dados na seção de referência do BASIC.

Comandos associados a TODOS os modos de tela

SCREEN Define o modo de tela.
CLS Limpa a tela.
COLOR Define as cores de texto, de fundo e das margens.

Comandos e funções associados a MODOS DE TEXTO SOMENTE

WIDTH Define a largura do modo de tela de texto.
LOCATE Define a posição do cursor no modo de tela de texto.

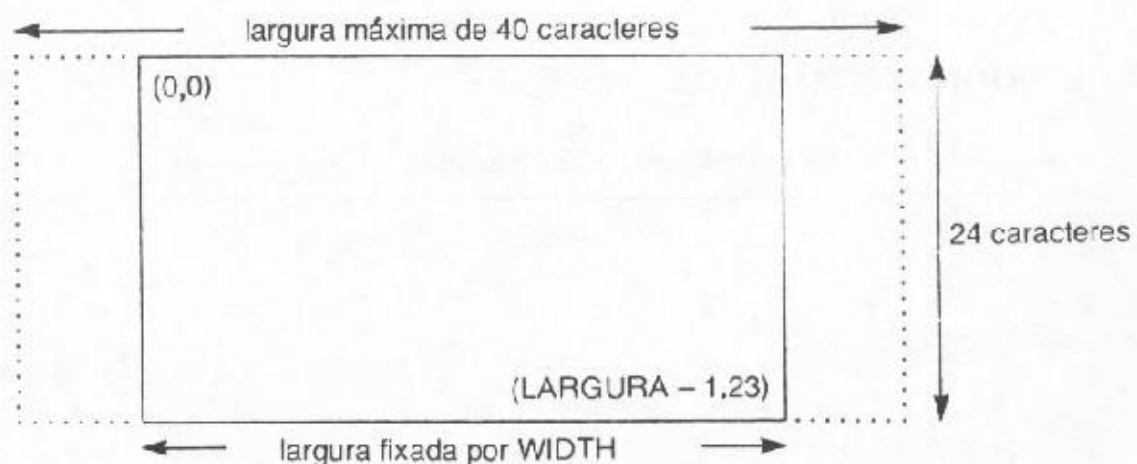
TAB	Define a posição horizontal do cursor na linha atual.
CSRLIN	Retorna a posição vertical atual do cursor.
POS(0)	Retorna a posição horizontal atual do cursor.

Comandos e funções associados a MODOS GRÁFICOS SOMENTE

CIRCLE	Desenha um círculo.
DRAW	Desenha de acordo com a macrolinguagem macro.
LINE	Desenha linhas, quadrados e retângulos.
PAINT	Pinta com a cor de texto atual.
PSET	Desenha um ponto.
PRESET	Elimina um ponto.
POINT	Retorna a cor do ponto referido.

Comandos associados ao PROCESSADOR DE EXIBIÇÃO DO VÍDEO (VDP)

VPOKE	Executa uma sentença POKE na RAM de vídeo.
VPEEK	Executa uma sentença PEEK na RAM de vídeo.
VDP	Retorna valores do registrador VDP.
BASE	Retorna o endereço base das tabelas das RAM de vídeo.

MODO 0: MODO DE TEXTO 40 × 24

O MODO 0 dá a você 40 caracteres por linha, que é o número máximo de caracteres que você pode ter em uma linha em computador MSX. Nesse modo você pode

escrever e editar o seu programa. Você descobrirá que a exibição de um programa neste modo é mais fácil de se ler.

Entretanto, o MODO 0 tem algumas desvantagens. Por exemplo, os caracteres são exibidos no formato comprimido, isto é, 6 × 8 em vez de 8 × 8 e, dessa forma, alguns caracteres gráficos irão aparecer cortados. Os dois pontos mais à direita em cada caractere não são exibidos! Entretanto, isto não afeta caracteres alfanuméricos.

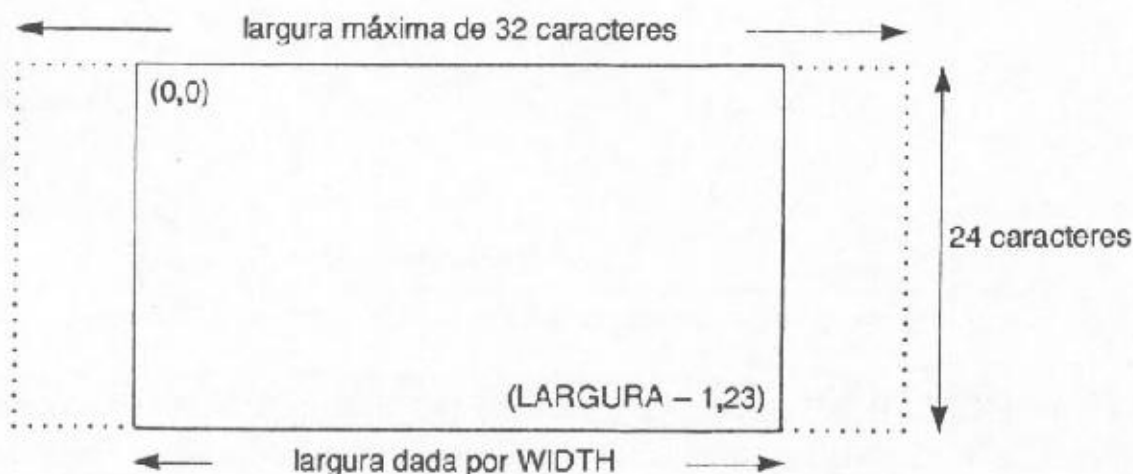
A largura padrão do modo de tela no MODO 0 é de 37 caracteres. Você pode aumentar a largura da exibição para 40 caracteres, usando o comando WIDTH. As coordenadas do canto superior esquerdo são (0,0), enquanto que o canto inferior direito é (WIDTH 1,23), ou quando inicializado é (38,23).

Você pode posicionar o cursor de texto usando TAB e LOCATE. Para descobrir onde está o cursor, são usadas as funções POS(0) e CSRLIN.

No MODO 0, você não pode usar SPRITES e também só pode usar duas das 16 cores, apesar do fato da combinação de duas cores ser de sua livre escolha. As cores padrões são as mesmas do MODO 1: texto em branco e fundo azul. Observe que o MODO 0 não tem qualquer margem: ela simplesmente desaparece da exibição, de modo que definir cores no MODO 0 é completamente inútil. Observe que as cores da exibição irão mudar imediatamente após a execução do comando COLOR.

Apesar de ser modo gráfico, você está livre para usar comandos INPUT e a lista das teclas de funções é exibida na linha 23, a menos que seja desativada por KEY OFF.

Todos os comandos gráficos e funções deste modo serão tratados como "Illegal function call", de modo que vamos examiná-las.

MODO 1: MODO TEXTO 32 × 24

O MODO 1 tem uma resolução de 32 × 24 caracteres, mas não é gráfico. Neste modo você pode escrever e ditar programas. Ele pode exibir 8 × 8 caracteres sem qualquer corte (como no caso do MODO 0).

Você pode aumentar a largura de exibição para 32 caracteres usando o comando WIDTH. O motivo para a largura padrão ser menor que 32 é que algumas TVs e monitores não podem exibir o modo de tela inteiro.

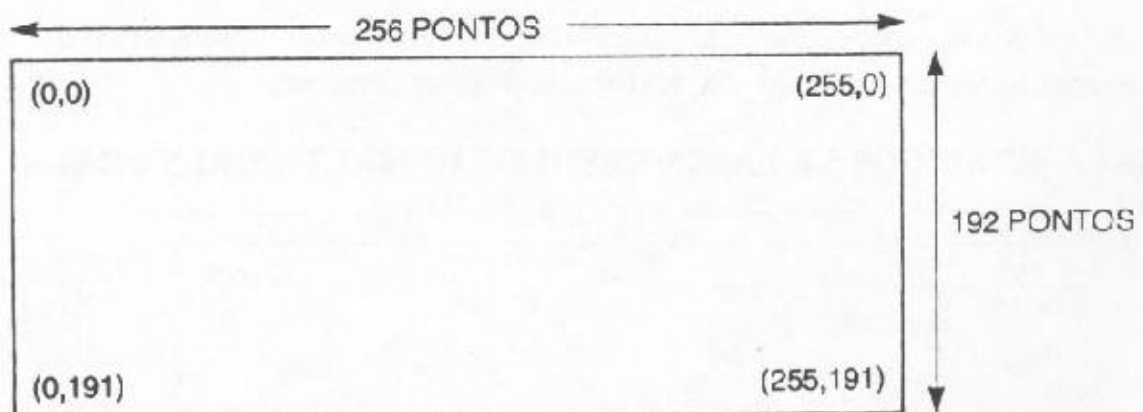
A coordenada do canto superior esquerdo é (0,0), enquanto que a do canto inferior direito é (WIDTH–1,23).

Você pode posicionar o cursor de texto usando TAB e LOCATE. Para descobrir onde está o cursor, use as funções POS(0) e CSRLIN.

Você pode usar somente 2 das 16 cores, apesar delas serem de sua inteira escolha. Observe que as cores da exibição mudam imediatamente após a execução do comando COLOR.

Apesar de ser modo gráfico, você está livre para usar o comando INPUT, e a lista das teclas de função é exibida na linha 23 a menos que seja desativada por KEY OFF.

Todos os comandos gráficos e funções neste modo, exceto aquelas envolvidas com sprites, serão tratadas como chamadas ilegais de função.

MODO 2: MODO GRÁFICO DE ALTA RESOLUÇÃO DE 256 × 192:

O MODO 2 é o modo gráfico mais largamente usado, pois fornece ao usuário um modo gráfico de tela de alta resolução com 16 cores. Neste modo você pode usar

sprites e a macrolinguagem gráfica através do comando DRAW. Este é o modo usado na maioria dos programas de jogos.

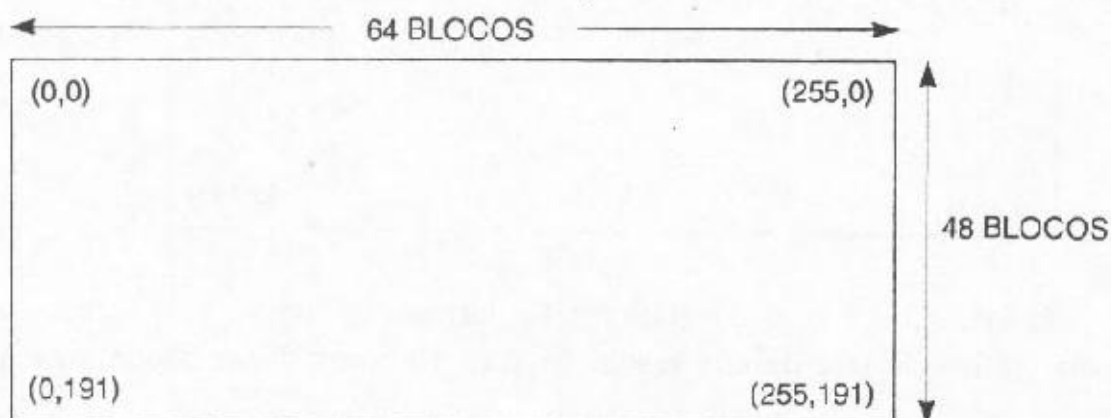
Sua resolução horizontal é de 256 pontos, enquanto que a resolução vertical é de 192 pontos. Entretanto, a resolução de cores é um tanto diferente: é de 32 × 192. Isto significa que você somente pode selecionar duas cores por bloco horizontal de 8 × 1 ponto. Você pode selecionar a cor de fundo e a de texto para cada bloco de 8 pontos, mas se desenhar um ponto com uma terceira cor, os pontos desenhados na cor anterior dentro desse bloco irão alterar-se para a nova cor de fundo automaticamente. Devido a isso, você pode obter figuras borradas se não for cuidadoso com o local onde está desenhando e com que cor. Entretanto, se você desenhar cuidadosamente com a resolução de 8 cores, poderá produzir belas figuras. Mais detalhes sobre as técnicas de desenho avançado serão fornecidas mais adiante.

Você pode usar sprites neste modo. Os sprites estão em um plano do modo de tela diferente do plano do modo de tela principal. Isto significa que não importa onde os sprites sejam exibidos; eles não afetam o que é desenhado no modo de tela principal de nenhum modo. Você irá ver que quando um sprite ocupa uma parte do modo de tela em particular, o modo de tela principal continua totalmente inalterado.

No modo gráfico 2, a lista das teclas de função não pode ser exibida. Você também irá descobrir que o comando PRINT não funciona neste modo. Para imprimir no modo gráfico de tela você deve abrir um arquivo no modo de tela usando PRINT#. Isto será explicado com mais detalhes na seção de impressão em modo gráfico.

O comando COLOR não altera as cores imediatamente neste modo. Para alterar as cores de um modo de tela inteiro você deve executar um comando CLS.

Você não pode usar o comando INPUT neste modo, porque ele força o modo de tela a retornar para o modo de texto usado anteriormente.

MODO 3: GRÁFICOS DE BAIXA RESOLUÇÃO MULTICORES DE 64 × 48

O MODO 3, Multicolorido, dá uma resolução baixa de 64 por 48 blocos, mas os blocos individuais podem ter suas próprias cores (ao contrário do MODO 2). Você não irá obter efeitos de borrões como no MODO 2.

Suas coordenadas são as mesmas do MODO 2; mas grupos de 4 por 4 pontos, representam um bloco. Isto permite que você desenhe nas mesmas coordenadas como no MODO 2, o que é vantajoso.

Você pode usar sprites neste modo da mesma maneira que no MODO 2.

No MODO 3, a lista das teclas de função não é exibida. Você também descobrirá que o comando PRINT não funciona neste modo. Para imprimir no modo gráfico de tela, você deve primeiro abrir um arquivo no modo de tela e usar PRINT#. O tamanho dos caracteres impressos será 4 vezes maior que no MODO 1 e 2. Isso será explicado com mais detalhes na seção de impressão em modo gráfico.

O comando COLOR não altera a cor imediatamente neste modo. Para alterar a cor do modo de tela inteiro, você deve usar o comando CLS.

Você não pode usar o comando INPUT neste modo pois iria forçar o modo de tela de volta para o modo de tela usado anteriormente.

CAPÍTULO 41

GRÁFICOS AVANÇADOS II

CORES NO MODO 2 DE ALTA RESOLUÇÃO

Este capítulo trata exclusivamente da maneira como as cores são usadas no MODO 2. O comando COLOR define a cor para o fundo, do texto e das margens atuais. Entretanto, a cor de fundo do modo de tela inteiro não irá alterar-se a menos que você o limpe com um comando CLS. Após o comando COLOR ter sido executado, qualquer ponto, linha ou figura desenhada será na nova cor de texto, a menos que outra cor seja especificada dentro de comandos de desenho.

A resolução de cor do MODO 2 é de 32 × 192 pontos. Isto significa que você pode selecionar duas cores por bloco horizontal de 8 × 1 ponto. Você pode especificar as cores de fundo e de texto para cada bloco de 8 pontos mas, se você tentar desenhar um ponto com uma terceira cor sobre um ponto que já tenha duas cores, os pontos na cor de texto dentro desse bloco irão alterar-se para essa nova cor de texto automaticamente, podendo causar efeitos de borrões se você não for cuidadoso. O pequeno programa abaixo demonstra isso:

```
10 SCREEN 2
20 COLOR 4,15,15
30 CLS
40 LINE (0,0)-(4,191),4,BF
50 IF NOT STRIG(0) THEN 50
60 LINE (6,0)-(6,191),10
70 GOTO 70
```

LINHA 10	SELECIONA MODO 2
LINHA 20	DEFINE O FUNDO COMO BRANCO
LINHA 30	LIMPA O MODO DE TELA DANDO A COR BRANCA
LINHA 40	DESENHA UM RETÂNGULO EM AZUL
LINHA 50	ESPERA ATÉ QUE A BARRA DE ESPAÇOS SEJA PRESSIONADA
LINHA 60	DESENHA UMA LINHA EM AMARELO-ESCURO JUNTO AO RETÂNGULO AZUL
LINHA 70	FICA NO MODO GRÁFICO DE TELA

Execute este programa. Em primeiro lugar, você verá um retângulo azul desenhado sobre o fundo branco, no centro do modo de tela. Então, se você pressionar a barra de espaço <espaços>, verá uma linha amarela sendo desenhada e o retângulo azul ficará amarelo. Por que a cor do retângulo azul mudou ao desenharmos uma linha amarela ao redor dele?

Para entender o que aconteceu, vamos ver como os atributos de cores são armazenados na RAM do vídeo (VRAM).

Se você pegar a seção do canto superior esquerdo do modo de tela e ampliá-la, ela pode ser representada como se fosse a tabela a seguir (A significa azul e B significa branco):

Y X =	0	1	2	3	4	5	6	7	8	9	.	.
0	A	A	A	A	B	B	B	B	B	B		
1	A	A	A	A	B	B	B	B	B	B		
2	A	A	A	A	B	B	B	B	B	B		
3	A	A	A	A	B	B	B	B	B	B		
4												

Dentro da RAM do vídeo do MSX, a figura acima está armazenada em código binário como mostrado a seguir. Cada bit corresponde a um ponto na tela, 1 = cor de texto e 0 = cor de fundo. Neste caso, 1 representa o azul e 0 o branco.

	0	1	2	3	4	5	6	7	8	9	.	.	.
0	1	1	1	1	0	0	0	0	0	0	0	0	
1	1	1	1	1	0	0	0	0	0	0	0	0	
2	1	1	1	1	0	0	0	0	0	0	0	0	
3	1	1	1	1	0	0	0	0	0	0	0	0	
4													

Entretanto, esta tabela de memória não revela a cor de cada ponto. Isto porque existe uma tabela separada de atributo de cor na VRAM que contém as cores de fundo e de texto para cada bloco de 8 × 1 ponto. Vamos pegar o bloco do canto superior esquerdo e ver como o seu padrão de cores é armazenado na memória.

TABELA DE PADRÕES

0	1	2	3	4	5	6	7
1	1	1	1	0	0	0	0

TABELA DE ATRIBUTOS DE COR

1	0
4	15

Como você pode ver, enquanto a tabela de padrão informa que o bit está na cor de fundo ou de texto, a tabela de atributo informa a você qual é a cor de fundo e a qual de texto. Isto significa que é fisicamente impossível colocar mais de duas cores por bloco. Desse modo, quando você tentar desenhar uma linha amarela na coluna 6, em primeiro lugar a cor de texto na tabela de atributo será alterada para o código de cor do amarelo, isto é, 10, e então o ponto 6 será definido para a nova cor de texto:

TABELA DE PADRÕES

0	1	2	3	4	5	6	7
1	1	1	1	0	0	1	0

TABELA DE ATRIBUTOS DE COR

1	0
10	15

Os pontos 0, 1, 2 e 3 permanecem na mesma cor de texto, mas agora a própria cor de texto foi alterada do azul para o amarelo, e por isso a cor dos pontos 0, 1, 2 e 3 também mudou.

A explicação é simples quando você a vê em termos de tabela de memória, não é mesmo?

Desse modo, o que devemos fazer para resolver esse problema? Existe somente uma resposta: não usar mais de duas cores em um bloco.

Por exemplo, o programa anterior pode ser alterado de modo que a linha amarela não seja desenhada no mesmo bloco do retângulo azul, movendo a figura inteira para a

esquerda em dois pontos. O programa melhorado, que desenhará um retângulo azul e uma linha amarela em seus lados, se parecerá com o seguinte:

```
10 SCREEN 2
20 COLOR 4,15,15
30 CLS
40 LINE (2,0)-(6,191),4,BF
50 IF NOT STRIG(0) THEN 50
60 LINE (8,0)-(8,191),10
70 GOTO 70
```

LINHA 10	SELECIONA O MODO 2
LINHA 20	DEFINE A COR DE FUNDO PARA BRANCO
LINHA 30	LIMPA O MODO DE TELA E COLOCA A COR BRANCA
LINHA 40	DESENHA UM RETÂNGULO EM AZUL
LINHA 50	ESPERA ATÉ QUE A BARRA DE ESPAÇOS SEJA PRESSIONADA
LINHA 60	DESENHA UMA LINHA EM AMARELO-ESCURO AO REDOR DO RETÂNGULO AZUL
LINHA 70	MANTÉM O MODO GRÁFICO DE TELA

Observe as alterações feitas nos especificadores de coordenadas nas linhas 40 a 60.

Nota: Se você quiser saber mais sobre como o modo de tela é armazenado na memória RAM de vídeo, vá para a seção de RAM de vídeo.

CAPÍTULO 42

GRÁFICOS AVANÇADOS III

COMO IMPRIMIR NO MODO GRÁFICO DE TELA USANDO ARQUIVOS

Imprimir no modo gráfico de tela não é exatamente a mesma coisa que imprimir no modo de texto. Você deve abrir um arquivo no modo gráfico de tela e então usar o comando PRINT#. De fato, você pode abrir um arquivo para um modo de tela de texto se quiser, mas isso não é necessário.

Eis um programa de demonstração que imprimirá em quatro modos de tela, usando arquivos. A linha 30 abre os arquivos em função do modo (M) de tela selecionado. Ela executa OPEN "CRT:" AS#1 quando M for menor que 2. Isto significa abrir um arquivo para um modo de tela de texto ("CRT:" AS, arquivo número #1). Quando M for maior ou igual a 2, então o comando executa OPEN "GRP:" AS#1 que significa abrir um arquivo para o modo gráfico de tela. Um arquivo aberto deve ser fechado assim que a sua função tiver terminado. Isto é feito na linha 60: CLOSE# 1.

```
10 FOR M=0 TO3
20 SCREEN M
30 IF M<2 THEN OPEN"crt:"AS #1 ELSE OPEN "grp:" AS #1
40 PRINT#1,"este e o modo ";M
50 FOR G=1 TO 1000:NEXT G
60 CLOSE#1
70 NEXT M
```

LINHA 10	LOOP DE 0 A 3
LINHA 20	SELECIONA O MODO DE TELA
LINHA 30	SE MODO TEXTO, ENTÃO ABRA MODO DE TELA GRÁFICO "CRT:"; SENÃO ABRA "GRP:"
LINHA 40	IMPRIME UMA MENSAGEM
LINHA 50	ESPERA
LINHA 60	FECHA O ARQUIVO
LINHA 70	PRÓXIMO MODO

Neste programa, você verá o modo de tela mudar para cada modo diferente enquanto imprime a mensagem "ESTE É O MODO...". Observe que no modo de tela de alta resolução a impressão é basicamente no mesmo formato que no MODO 1, isto é, 32 × 24 caracteres. No MODO 3, o texto tem quatro vezes o tamanho normal do modo texto. Isto porque o modo multicores é de baixa resolução, e para se imprimir caracteres ele precisa fazer isso em blocos gráficos (de 4 × 4 pontos).

COMO ESCREVER NO MODO 2 DE ALTA RESOLUÇÃO

Quando se imprime um texto no MODO 2 você não pode usar o comando LOCATE para posicionar os caracteres a serem impressos. Isto se deve ao fato de o computador não usar o cursor de texto no modo gráfico. Em vez disso, ele usa o cursor gráfico e, dessa forma, o computador imprime o texto na última posição do cursor gráfico. Para posicionar os caracteres a serem impressos, use o comando DRAW.

Programa exemplo: PRINT# e PRINT# USING no modo gráfico 2

```
10 SCREEN 2
20 OPEN "GRP:" AS #1
30 DRAW "BM100,100"
40 PRINT#1,"SOCORRO"
50 DRAW "BM102,102"
60 PRINT#1,"SOCORRO"
70 DRAW "BM100,120"
80 PRINT#1,USING"###.######";ATN(1)*4
90 GOTO 90
```

LINHA 10 SELECIONA O MODO DE TELA 2
LINHA 20 ABRE O ARQUIVO PARA GRP
LINHA 30 POSICIONA O CURSOR GRÁFICO E IMPRIME AÍ
LINHA 50 REPOSICIONA O CURSOR
LINHA 60 IMPRIME SOCORRO NOVAMENTE
LINHA 70 REPOSICIONA O CURSOR OUTRA VEZ
LINHA 80 PRINT# USING
LINHA 90 MANTÉM O MODO GRÁFICO

No exemplo anterior você verá duas mensagens "SOCORRO", uma em cima da outra. Isto acontece porque, no modo gráfico, o computador não apaga antes de imprimir, podendo criar alguns efeitos interessantes, mas freqüentemente é um aborrecimento. A única maneira de apagar o que está impresso em uma parte do modo de tela é usar o comando LINE e escrever em cima com um retângulo preenchido na cor de fundo.

Se você incluir a linha:

```
45    LINE (100,100)–STEP(43,8),4,BF
```

apagará a mensagem "SOCORRO" escrita na linha 40, evitando a sobreposição.

Como pôde ver no programa anterior, você também pode usar PRINT USING no formato PRINT# USING. As várias sintaxes de PRINT USING são as mesmas para os diversos modos gráficos, bem como para os modos texto.

COMO IMPRIMIR COM CORES NO MODO 2 GRÁFICO

Uma das vantagens de se imprimir em um modo gráfico é a de se poder usar todas as 16 cores para imprimir caracteres, bem como gráficos. Para imprimir na cor especificada, você tem de especificar a cor de texto antes de usar o PRINT#.

Eis um exemplo para ilustrar a impressão em 16 cores:

```
10 COLOR 1,15,15
20 SCREEN 2
30 OPEN "grp:" AS #1
40 FOR I=0 TO 15
50 COLORI
```

```
60 DRAW"bm+8,0"
70 PRINT#1,"Cor codigo";I
80 NEXTI
90 GOTO 90
```

LINHA 10	A COR DE FUNDO É BRANCA
LINHA 20	MODO DE TELA 2
LINHA 30	ABRE O ARQUIVO PARA GRP COMO # 1
LINHA 40	LOOP DE 0 A 15
LINHA 50	COR DO LOOP
LINHA 60	MOVE O CURSOR GRÁFICO
LINHA 70	MENSAGEM
LINHA 80	PRÓXIMO LOOP
LINHA 90	MANTÉM O MODO DE TELA

Este programa produzirá a seguinte exibição:

CÓDIGO DE COR 0	TRANSPARENTE
CÓDIGO DE COR 1	PRETO
CÓDIGO DE COR 2	VERDE (MÉDIO)
CÓDIGO DE COR 3	VERDE (CLARO)
CÓDIGO DE COR 4	AZUL (ESCURO)
CÓDIGO DE COR 5	AZUL (CLARO)
CÓDIGO DE COR 6	VERMELHO (ESCURO)
CÓDIGO DE COR 7	CIAN
CÓDIGO DE COR 8	VERMELHO (MÉDIO)
CÓDIGO DE COR 9	VERMELHO (CLARO)
CÓDIGO DE COR 10	AMARELO (ESCURO)
CÓDIGO DE COR 11	AMARELO (CLARO)
CÓDIGO DE COR 12	VERDE (ESCURO)
CÓDIGO DE COR 13	MAGENTA
CÓDIGO DE COR 14	CINZA
CÓDIGO DE COR 15	BRANCO

COMO IMPRIMIR NO MODO GRÁFICO 3 MULTICORES

No MODO 3 Multicores, o texto tem quatro vezes o tamanho do modo normal porque a exibição gráfica é de baixa resolução. Por isso, para imprimir caracteres gráficos,

devem ser usados blocos gráficos, e os caracteres resultantes são enormes. Eis um programa de demonstração:

```
10 SCREEN 3
20 OPEN "GRP:" AS #1
30 DRAW "BM0,0"
40 PRINT#1,"PI"
50 DRAW "BM0,65"
60 PRINT#1,USING "#.######";ATN(1)*4
70 GOTO 70
```

LINHA 10	SELECIONA O MODO MULTICORES
LINHA 20	ABRE UM ARQUIVO PARA O MODO GRÁFICO DE TELA
LINHA 30	POSICIONA O CURSOR
LINHA 40	IMPRIME "PI"
LINHA 50	REPOSICIONA O CURSOR
LINHA 60	IMPRIME O VALOR DE PI
LINHA 70	MANTÉM O MODO GRÁFICO DE TELA

CAPÍTULO **43**

GRÁFICOS AVANÇADOS IV

OS SPRITES

O processador de exibição de vídeo do MSX tem a facilidade de produzir sprites. Sprites são caracteres ou figuras definidas pelo usuário, que podem ser exibidos no modo de tela sem afetar o modo gráfico de tela de fundo. Isto porque os sprites são exibidos em planos de modo de tela diferentes. Eles podem ser escondidos uns atrás dos outros e podem ser movidos sem provocar confusão. Eles dão a você o poder de criar jogos e animações sem muitos problemas com os gráficos.

TAMANHO DOS SPRITES

Existem quatro tamanhos de sprites que você pode utilizar, sendo somente um de cada vez. O tamanho é determinado usando-se o comando SCREEN.

COMANDO	TAMANHO	
SCREEN,0	8 por 8	ponto não ampliado
SCREEN,1	8 por 8	ponto ampliado para 16 × 16
SCREEN,2	16 por 16	ponto não ampliado
SCREEN,3	16 por 16	ponto ampliado para 32 × 32

Observe que a ampliação fornece uma resolução de 2×2 pontos.

DEFININDO SPRITES: SPRITE$

Você pode definir os seus sprites usando SPRITE$. Você pode ter até 256 padrões de sprites quando estiver usando os tamanhos de sprites 0 ou 1 (8 × 8 pontos), ou 64 nos tamanhos 2 ou 3 (16 × 16 pontos).

Sintaxe

Sprite$ (<inteiro>)=<string>

<inteiro> = número de padrão do sprite
<inteiro> = 0 a 255 quando o sprite for 0 ou 1
<inteiro> = 0 a 63 quando o sprite for 2 ou 3

SPRITE$ é uma string. O tamanho da string sprite é fixado em 32 bytes (ou caracteres); para sprites pequenos (8×8) você somente precisa definir os primeiros 8 bytes.

Cada byte na string representa uma linha de 8 pontos, e é somada na string do sprite incluindo um caractere com o código ASCII igual ao do byte. Isso pode ser feito simplesmente usando-se a função CHR$. Por exemplo, para definir um sprite de 8 × 8 pontos:

```
SPRITE$ (<inteiro>)=CHR$(<prim. linha>)+CHR$(<seg. linha>)+CHR$(<terc.
linha>)+CHR$(<qua. linha>)+CHR$(<qui. linha>)+CHR$(<sex.
linha>)+CHR$(<set. linha>)+CHR$(<oit. linha>)
```

A função CHR$( ) pode ser substituída pelo caractere atual se você souber qual é.

Para definir um sprite de 16 × 16, você deve definir todos os 32 bytes do padrão do sprite.

Eis um bom modo para definir um sprite de 8 × 8 pontos.

Em primeiro lugar desenhe o seu sprite em um papel.

128	64	32	16	8	4	2	1		BINARIO		DECIMAL
·	·	·	1	·	·	·	·	=	&B00010000	=	16
·	·	1	1	·	·	·	·	=	&B00110000	=	48
·	1	1	1	·	·	·	·	=	&B01110000	=	112
1	1	1	1	1	1	1	1	=	&B11111111	=	255
1	1	1	1	1	1	1	1	=	&B11111111	=	255
·	1	1	1	·	·	·	·	=	&B01110000	=	112
·	·	1	1	·	·	·	·	=	&B00110000	=	48
·	·	·	1	·	·	·	·	=	&B00010000	=	16

Desta forma, você pode formar uma expressão de strings como a seguinte:

```
SPRITE$(0)=CHR$(16)+CHR$(48)+CHR$(112)+CHR$(255)+CHR$(255)+
CHR$(112)+CHR$(48)+CHR$(16)
```

Exemplo:

Usar números binários é o melhor caminho para definir um sprite, pois é fácil ver como o sprite vai aparecer no programa.

```
10 SCREEN 2,0
20 FOR I=1 TO 8
30 READ A$
40 S$=S$+CHR$(VAL("&B"+A$))
50 NEXT I
60 SPRITE$(0)=S$
70 PUT SPRITE 0,(100,100),15,0
80 GOTO 80
90 REM DADOS BINARIOS
100 DATA 00010000
110 DATA 00110000
120 DATA 01110000
130 DATA 11111111
140 DATA 11111111
150 DATA 01110000
160 DATA 00110000
170 DATA 00010000
```

LINHA 10	EMPREGA O MODO DE TELA DE ALTA RESOLUÇÃO, SPRITE DE TAMANHO NORMAL
LINHA 30	LÊ OS NÚMEROS BINÁRIOS COMO STRING
LINHA 40	S$ É UMA STRING TEMPORÁRIA
LINHA 60	DEFINE O SPRITE 0
LINHA 70	COLOCA O SPRITE O EM x=100,y=100, COR BRANCA, NO PLANO 0 DO SPRITE

Resultado: você vê uma flecha no meio do modo de tela.

Vamos tratar mais adiante do comando **PUT SPRITE** detalhadamente.

Se você acha "chato" calcular os valores decimais de cada byte, então o programa pode ser escrito em uma versão mais curta. O programa acima pode ser reduzido às seguintes poucas linhas:

```
10 SCREEN 2,0
20 SPRITE$(0)=CHR$(16)+CHR$(48)+CHR$(112)+CHR$(255)+CH
R$(112)+CHR$(48)+CHR$(16)
70 PUT SPRITE 0,(100,100),15,0
80 GOTO 80
```

LINHA 10	EMPREGA O MODO DE TELA DE ALTA RESOLUÇÃO, SPRITE DE TAMANHO NORMAL
LINHA 20	DEFINE O SPRITE 0
LINHA 70	COLOCA O SPRITE 0 EM x=100,y=100, COR BRANCA NO PLANO 0 DO SPRITE

Como definir um sprite de 16 × 16

```
128 64 32 16  8  4  2  1128 64 32 16  8  4  2  1
 .   .  .  1  1  1  1  1  1  1  1  1  1  .  .  .
 .   .  1  1  1  1  1  1  1  1  1  1  1  1  .  .
 .   1  1  .  .  .  .  .  .  .  .  .  .  1  1  .
 1   1  .  .  .  1  1  1  1  1  1  .  .  .  1  1
 1   1  .  .  .  1  1  1  1  1  1  .  .  .  1  1
 1   1  .  .  1  .  .  .  .  .  .  1  .  .  1  1
 .   1  1  .  1  .  .  .  .  .  .  1  .  1  1  .
 .   1  1  .  .  1  .  .  .  .  1  .  .  1  1  .
 .   .  1  1  .  .  1  1  1  1  .  .  1  1  .  .
 .   .  1  1  .  1  .  .  .  .  1  .  1  1  .  .
 .   .  .  1  1  .  1  1  1  1  .  1  1  .  .  .
 .   .  .  1  1  .  .  .  .  .  .  1  1  .  .  .
 .   .  .  .  1  1  .  .  .  .  1  1  .  .  .  .
 .   .  .  .  1  1  .  .  .  .  1  1  .  .  .  .
 .   .  .  .  .  1  1  .  .  1  1  .  .  .  .  .
 .   .  .  .  .  .  1  1  1  1  .  .  .  .  .  .
 .   .  .  .  .  .  .  1  1  .  .  .  .  .  .  .
```

Eis o diagrama em binário e decimal:

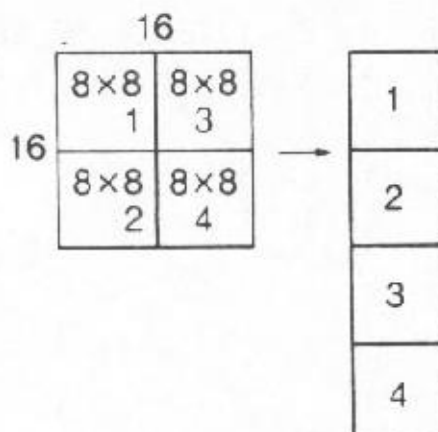
```
00011111 =  31 11111000 = 248
00111111 =  63 11111100 = 252
01100000 =  96 00000110 =   6
11000111 = 199 11100011 = 227
11001000 = 200 00010011 =  19
01101000 = 104 00010110 =  22
01100100 = 100 00100110 =  38
00110011 =  51 11001100 = 204

00110100 =  52 00101100 =  44
00011011 =  27 11011000 = 216
00011000 =  24 00011000 =  24
00001100 =  12 00110000 =  48
00001100 =  12 00110000 =  48
00000110 =   6 01100000 =  96
00000011 =   3 11000000 = 192
00000001 =   1 10000000 = 128
```

Os padrões de SPRITE 16 × 1 estão armazenados na RAM de vídeo da seguinte maneira:

16

8×8 1	8×8 3
8×8 2	8×8 4

16 → 1 / 2 / 3 / 4

Um SPRITE$ 16 × 16 = "Sprite do quadrante 1" + "Sprite do quadrante 2" + "Sprite do quadrante 3" + "Sprite do quadrante 4".

Eis um programa razoavelmente eficiente que define e exibe o sprite de 16 × 16 que projetamos anteriormente. Os dados estão armazenados nos DATA e são incluídos um a um no SPRITE$ no loop FOR/TO/NEXT.

```
10 SCREEN 2,2
20 FOR I=1 TO 32
30 READ B%
40 S$=S$+CHR$(B%)
50 NEXT I
60 SPRITE$(0)=S$
70 PUT SPRITE 0,(100,100),15,0
80 GOTO 80
90 DATA 31,63,96,199,200,104,100,51
100 DATA 52,27,24,12,12,6,3,1
110 DATA 248,252,6,227,19,22,38,204
120 DATA 44,216,24,48,48,96,192,128
```

LINHA 10	EMPREGA O MODO DE TELA 2 COM TAMANHO DE SPRITE DE 16 POR 16
LINHA 20	LOOP PARA LER 32 BYTES
LINHA 30	LER
LINHA 40	INCLUI DADOS EM S$
LINHA 60	DEFINE A STRING DO SPRITE 0
LINHA 90	DADOS PARA O QUADRANTE 1
LINHA 100	DADOS PARA O QUADRANTE 2
LINHA 110	DADOS PARA O QUADRANTE 3
LINHA 120	DADOS PARA O QUADRANTE 4

COMO COLOCAR UM SPRITE NO MODO DE TELA: PUT SPRITE

Existem 32 planos de sprite em frente dos planos de texto/gráfico, com o plano sprite nº 0 na frente de todos. Isto significa que quanto menor o número do plano maior a prioridade. Se existirem dois sprites sobrepostos, aquele com o menor número de plano será exibido na frente, com o outro escondido atrás.

Você pode definir até 256 padrões de sprites, mas somente um sprite pode ser exibido em um plano em particular. Isso limita o número máximo de sprites exibidos ao mesmo tempo a 32.

Mais ainda: você só pode colocar um máximo de quatro sprites por linha horizontal.

Para colocar um sprite no modo de tela, use o comando PUT SPRITE. Ele tem a seguinte sintaxe:

PUT SPRITE <número do plano do sprite>[,<especificador de coordenada>][,<cor>][,
<número do sprite>]

O <número do plano do sprite> tem uma faixa de 0 a 31.

O <especificador de coordenada> diz ao computador onde colocar o canto superior esquerdo do sprite.

Existem dois formatos de <especificador de coordenadas>:

1. (<coordenada x>,<coordenada y>)
   Isso especifica uma posição absoluta no modo de tela.

2. STEP(<x>,<y>)
   Estas coordenadas são as coordenadas de um ponto relativas ao último ponto referenciado.

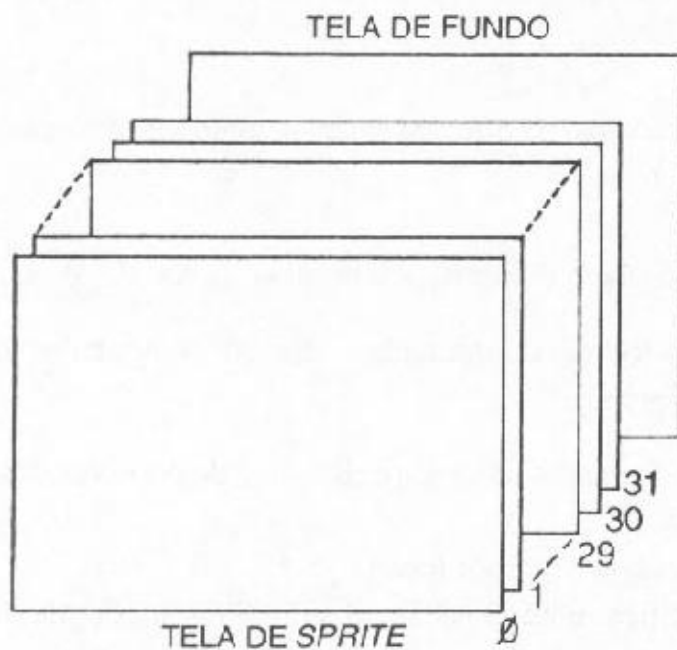
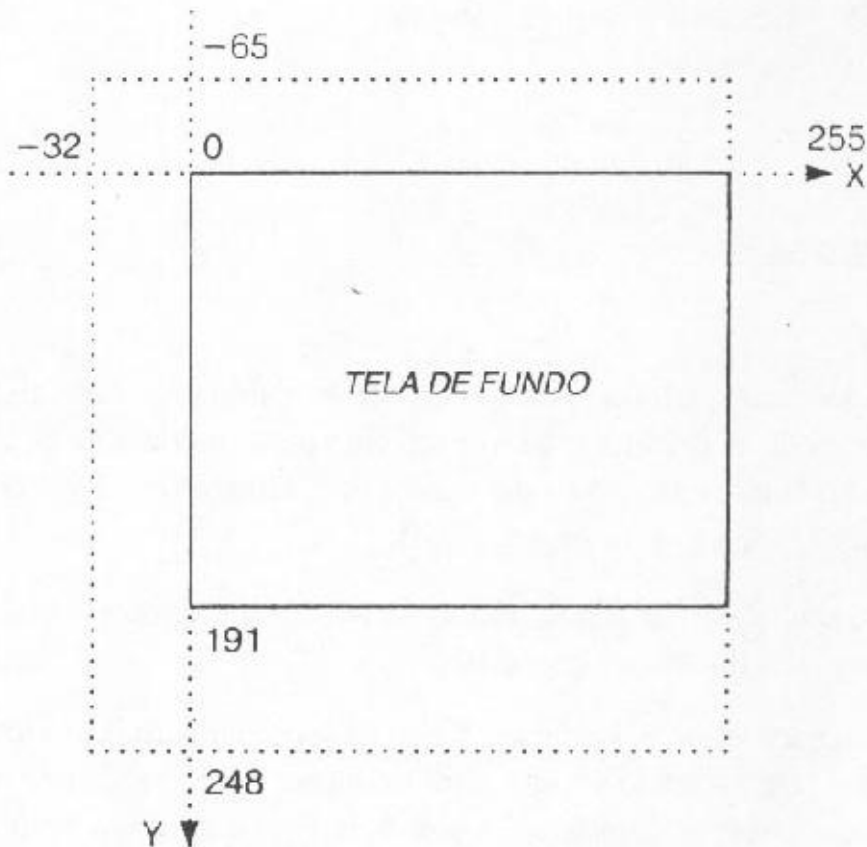
<x>,<y>, <coordenada x> e <coordenada y>, podem ser variáveis ou expressões, bem como simples constantes numéricas. Isto significa que os sprites podem ser controlados por variáveis permitindo a você movê-los suavemente.

Se o (<especificador de coordenada>) for omitido, então o computador colocará o SPRITE no último ponto referenciado.

Exemplos:

```
PUT SPRITE 0, (50,50),1,1
PUT SPRITE 1,STEP(10,10),12,2
```

O modo de tela de sprite é ligeiramente maior que o modo de tela de exibição normal. Ele tem uma faixa entre –32 e 255 na coordenada x e entre –65 e 248 na coordenada y. As faixas de coordenadas reais dos modos de telas são menores, de modo que, se o sprite for colocado fora do modo de tela de exibição real, ele será escondido parcial ou totalmente.

Os sprites têm a facilidade de unir os extremos do modo de tela. Isto significa que se um sprite sair fora do modo de tela, ele aparecerá outra vez no canto oposto da tela.

A tela de *sprite*:

COMO MOVER SPRITES

Alterando o valor do <especificador de coordenadas> no comando PUT SPRITE, você pode mover o sprite para qualquer lugar da tela. Quando o comando PUT SPRITE é executado, o sprite anterior na antiga localização do modo de tela é apagado automaticamente.

Este programa-exemplo exibe um quadrado movendo-se lentamente da direita para a esquerda, indo através do modo de tela para a esquerda e aparecendo pela direita.

```
10 SCREEN 2,0
20 SPRITE$(0)=STRING$(8,CHR$(255))
30 FOR X=200 TO -200 STEP -1
40 PUT SPRITE 0,(X,100),15,0
50 FOR D=1 TO 50:NEXT D
60 NEXT
70 END
```

LINHA 10 SELECIONA O MODO DE TELA, TAMANHO NORMAL
LINHA 20 DEFINE O SPRITE QUADRADO
LINHA 40 COLOCA O SPRITE EM X,100
LINHA 50 ESPERA

CORES DOS SPRITES

Você pode usar qualquer uma das 16 cores. Entretanto, pode usar somente uma cor por sprite, de modo que não é possível ter um sprite multicolorido. Para simular um sprite multicolorido, coloque dois ou mais sprites de diferentes cores em cima de outro em diferentes planos de sprites e mova-os juntos.

Observe que a cor de fundo de um sprite é sempre transparente, de modo que você pode ver através do vazio do sprite.

Este exemplo define dois sprites, 0 e 1. O sprite número 0 é exibido em branco e o sprite número 1 em amarelo-escuro. Na exibição você verá cada um dos sprites separadamente, bem como a combinação dos dois que simula um sprite multicolorido.

```
10 SCREEN 2,0
20 SPRITE$(0)=CHR$(16)+CHR$(48)+CHR$(112)+CHR$(255)+CH
R$(255)+CHR$(112)+CHR$(48)+CHR$(16)
30 SPRITE$(1)=CHR$(224)+CHR$(192)+CHR$(128)+CHR$(0)+CH
R$(0)+CHR$(128)+CHR$(192)+CHR$(224)
40 PUT SPRITE 0,(20,20),15,0
50 PUT SPRITE 1,(40,40),10,1
60 PUT SPRITE 2,(60,60),15,0
70 PUT SPRITE 3,(60,60),10,1
80 GOTO 80
```

LINHA 10 SELECIONA O MODO DE TELA, TAMANHO NORMAL
LINHA 20 DEFINE O SPRITE 0
LINHA 30 DEFINE O SPRITE 1
LINHA 40 COLOCA O SPRITE 0 EM BRANCO
LINHA 50 COLOCA O SPRITE 1 EM AMARELO-ESCURO
LINHA 40 COLOCA OS SPRITES 2 E 3 NAS MESMAS COORDENADAS
USANDO 2 CORES EM DIFERENTES PLANOS DE SPRITES

COMO ESCONDER SPRITES

Se você der um número especial à coordenada y, você pode desativar os sprites:

y = 208

Se for fornecido 208 (&HDO) à coordenada y, todos os planos de sprites maiores serão desativados até que um valor diferente de 208 seja dado a esse plano.

```
10 SCREEN 2,0
20 SPRITE$(0)=STRING$(8,CHR$(255))
30 PUT SPRITE 0,(20,20),15,0
40 PUT SPRITE 1,(40,40),15,0
50 PUT SPRITE 2,(60,208),15,0
60 PUT SPRITE 3,(80,80),15,0
70 PUT SPRITE 4,(100,100),15,0
80 GOTO 80
```

LINHA 10 SELECIONA O MODO DE TELA, TAMANHO NORMAL
LINHA 20 DEFINE UM SPRITE QUADRADO
LINHA 50 Y = 206 DESATIVA OS SPRITES DO PLANO 2 PARA A FRENTE: OS SPRITES DAS LINHAS 60 E 70 NÃO SÃO EXIBIDOS

Se for especificado y + 209 (&HDI) para y, então este sprite desaparece da tela.

```
10 SCREEN 2,0
20 SPRITE$(0)=STRING$(8,CHR$(255))
30 PUT SPRITE 0,(20,20),15,0
40 PUT SPRITE 1,(40,40),15,0
50 PUT SPRITE 2,(60,209),15,0
60 PUT SPRITE 3,(80,80),15,0
70 PUT SPRITE 4,(100,100),15,0
80 GOTO 80
```

LINHA 10 SELECIONA O MODO DE TELA, TAMANHO NORMAL
LINHA 20 DEFINE O SPRITE 0
LINHA 30 SPRITE QUADRADO NO PLANO 1
LINHA 40 SPRITE QUADRADO NO PLANO 2

LINHA 50 Y=209 DESATIVA ESTE SPRITE
LINHA 60 SPRITE QUADRADO NO PLANO 3
LINHA 70 SPRITE QUADRADO NO PLANO 4

A REGRA DO "QUINTO" SPRITE

Existe um limite máximo de quatro sprites que podem ser exibidos em uma linha horizontal. Se essa regra for violada, então somente os quatro sprites com os últimos números de planos de sprites serão exibidos normalmente. O restante não será exibido nessa linha. O número de sprite do quinto sprite, que violou a regra, pode ser encontrado no registrador de status VDP usando a função VDP (veja a seção da função VDP).

Exemplo:

```
10 SCREEN 2,0
20 SPRITE$(0)=STRING$(8,CHR$(255))
30 PUT SPRITE 0,(20,100),15,0
40 PUT SPRITE 1,(40,100),15,0
50 PUT SPRITE 2,(60,100),15,0
60 PUT SPRITE 3,(80,100),15,0
70 PUT SPRITE 4,(100,104),15,0
80 GOTO 80
```

LINHA 10 SELECIONA O MODO DE TELA EM TAMANHO NORMAL
LINHA 20 DEFINE O SPRITE 0
LINHA 30 SPRITE QUADRADO NA LINHA 100
LINHA 40 SPRITE QUADRADO NA LINHA 100
LINHA 50 SPRITE QUADRADO NA LINHA 100
LINHA 60 SPRITE QUADRADO NA LINHA 100
LINHA 70 SPRITE QUADRADO NA LINHA 100

No exemplo anterior, você verá quatro sprites quadrados, na mesma linha horizontal, a linha 100. O quinto sprite está na linha 104 e dessa forma ele mostra somente metade do quadrado, a outra metade sendo escondida devido à regra do quinto sprite. Se ele estivesse na linha 108, todo o quinto sprite seria visível.

ANIMAÇÃO DE SPRITES

Uma animação simples de sprites pode ser conseguida no MSX com um mínimo de trabalho. Tudo o que você tem a fazer é trocar rapidamente dois sprites de modo que os desenhos pareçam estar sendo trocados.

Aqui está um programa simples que mostra um invasor espacial com movimentos de pernas. O SPRITE$(0) contém o invasor com as pernas fechadas e o SPRITE$(1) o invasor com as pernas abertas.

```
10 SCREEN 2,1
20 SPRITE$(0)=CHR$(60)+CHR$(126)+CHR$(129)+CHR$(219)+C
HR$(126)+CHR$(36)+CHR$(36)+CHR$(36)
30 SPRITE$(1)=CHR$(60)+CHR$(126)+CHR$(129)+CHR$(219)+C
HR$(126)+CHR$(36)+CHR$(60)+CHR$(129)
40 PUT SPRITE 0,(100,100),11,0
50 FOR I=1 TO 500:NEXT
60 PUT SPRITE 0,(100,100),11,1
70 FOR I=1 TO 500:NEXT
80 GOTO 40
```

LINHA 20	INVASOR COM AS PERNAS FECHADAS
LINHA 30	INVASOR COM AS PERNAS ABERTAS
LINHA 40	SPRITE 0 (PERNAS FECHADAS) AMARELO
LINHA 50	ESPERA
LINHA 60	SPRITE 1 (PERNAS ABERTAS) AMARELO
LINHA 70	ESPERA

Programa de Demonstração

Neste exemplo você verá dois planetas orbitando um sol. O sistema inteiro flutua a partir do canto superior esquerdo da tela.

```
10 SCREEN 2,0
20 COLOR 15,1,1
30 CLS
40 SPRITE$(0)=CHR$(126)+STRING$(6,CHR$(255))+CHR$(126)
50 SPRITE$(1)=STRING$(3,CHR$(0))+CHR$(24)+CHR$(24)+STR
ING$(3,CHR$(0))
```

```
60 FOR I=0 TO 6.28 STEP .2
70 X=X+1.5
80 Y=Y+1
90 X1=30*COS(I)
100 Y1=30*SIN(I)
110 X2=15*COS(I)
120 Y2=15*SIN(I)
130 PUT SPRITE 0,(X,Y),11,0
140 PUT SPRITE 1,(X1+X,Y1+Y),9,1
150 PUT SPRITE 2,(X2+X,Y2+Y),15,1
160 NEXT
170 GOTO 60
```

LINHA 10	EMPREGA A TELA EM TAMANHO NORMAL, MARGENS E
LINHA 20	FUNDO PRETO, TEXTO PLANO BRANCO
LINHA 30	LIMPA A TELA
LINHA 40	SPRITE 0 É O SOL
LINHA 50	SPRITE 1 É UM PLANETA
LINHA 60	LOOP
LINHA 70	MOVE O SOL COM X DE 1.5 EM 1.5
LINHA 80	MOVE O SOL COM Y DE 1 EM 1
LINHA 90	X1 = COORDENADA X PARA O PLANETA 1
LINHA 100	Y1 = COORDENADA Y PARA O PLANETA 1
LINHA 110	X2 = COORDENADA X PARA O PLANETA 2
LINHA 120	Y2 = COORDENADA Y PARA O PLANETA 2
LINHA 130	COLOQUE O SOL NA NOVA POSIÇÃO
LINHA 140	COLOQUE O PLANETA 1 NA NOVA POSIÇÃO
LINHA 150	COLOQUE O PLANETA 2 NA NOVA POSIÇÃO
LINHA 160	LOOP SEGUINTE
LINHA 170	LOOP OUTRA VEZ

COMO DETECTAR COLISÕES DE SPRITES

O VDP TMS 9918/9929 pode detectar colisões de sprites. Isso é útil para aqueles jogos que envolvem disparos ou evitar inimigos. Existem dois comandos BASIC que tratam da detecção de colisões de sprites – ON SPRITE GOSUB e SPRITE ON/OFF/STOP.

O comando ON SPRITE GOSUB define a sub-rotina de detecção de colisão de sprites. Ele diz ao computador que a sub-rotina de detecção de colisão de sprites está no número de linha especificado.

O comando SPRITE ON ativa a detecção de colisão, que desvia o programa para a sub-rotina especificada pelo comando ON SPRITE GOSUB quando dois sprites colidem. Você deve executar o comando SPRITE ON para iniciar a detecção.

Se um ponto dos dois sprites se sobrepuser, isso será considerado uma colisão, assim como pontos transparentes que se sobreponham.

Apesar do computador poder detectar uma colisão de sprites, ele não pode dizer a você quais sprites colidiram ou mesmo onde a colisão aconteceu. Você deve programar o computador para que ele faça isso.

Uma vez ocorrida a interrupção, isto é, dois sprites colidiram, um comando automático SPRITE STOP é executado, parando o computador, chamando a sub-rotina outra vez.

Após ter deixado a sub-rotina de colisão de sprites, o computador irá executar automaticamente SPRITE ON, ativando assim a interrupção, a menos que um SPRITE OFF tenha sido executado na sub-rotina.

O comando SPRITE OFF desativa completamente a detecção de colisões de sprites pelo computador.

Exemplo:

Neste exemplo, você verá dois sprites quadrados, um amarelo e outro branco, aproximando-se um do outro de lados opostos do modo de tela. Quando eles colidem, ON SPRITE GOSUB entrará em funcionamento e desviará o BASIC para a sub-rotina de colisão de sprites. O SPRITE OFF na rotina de interrupção de sprite evita qualquer detecção posterior de colisão de sprites. Isso foi feito porque os dois sprites ainda estão se sobrepondo após o computador ter executado a sub-rotina. Se não houvesse um SPRITE OFF aí, o computador iria ficar eternamente nesse loop na sub-rotina de colisão de sprites. (Tente executar esta rotina SPRITE OFF, e veja o que acontece.)

```
10 ON SPRITE GOSUB 120
20 SCREEN 2,0
30 SPRITE$(0)=STRING$(8,CHR$(255))
40 SPRITE$(1)=STRING$(8,CHR$(255))
50 FOR I=10 TO 240
```

```
60 SPRITE ON
70 PUT SPRITE 0,(I,100),11,0
80 PUT SPRITE 1,(250-I,100),15,0
90 NEXT I
100 GOTO 50
110 REM SPRITE-ROTINA DE COLISAO
120 SPRITE OFF
130 BEEP
140 RETURN
```

LINHA 10	DEFINE A SUB-ROTINA DE COLISAO PARA 110
LINHA 20	DEFINE O MODO GRÁFICO DE TELA
LINHA 30	O SPRITE 0 É UM QUADRADO
LINHA 40	O SPRITE 1 TAMBÉM
LINHA 50	ATIVA O DETECTOR DE COLISÃO DE SPRITES
LINHA 60	LOOP
LINHA 70	O SPRITE AMARELO SE MOVE A PARTIR DA DIREITA
LINHA 80	O SPRITE BRANCO SE MOVE A PARTIR DA ESQUERDA
LINHA 90	LOOP SEGUINTE
LINHA 100	DESVIA PARA A LINHA 50
LINHA 110	INIBE A DETECÇÃO (UMA VEZ É SUFICIENTE)
LINHA 120	EMITE SOM
LINHA 130	VOLTA PARA O LOOP

Notas Sobre Colisões de Sprites

O computador pesquisa uma colisão de sprite a cada vez que executa um comando. Dessa forma pode-se dizer que o computador pesquisa colisões de sprites em qualquer intervalo de tempo em particular.

A função VDP pode dar a você o status do indicador (flag) de colisão de sprite do registrador 8 do processador de vídeo (veja o capítulo seguinte).

CAPÍTULO 44

GRÁFICOS AVANÇADOS V

COMO ACESSAR O PROCESSADOR DE VÍDEO (VDP) FUNÇÃO VDP

O VDP TMS 9929A tem 8 registradores de gravação e 1 de leitura. Você pode acessar esses registradores do VDP a partir do BASIC, usando a função VDP.

Aos registradores de gravação, VDP(0) a VDP(7), você somente pode atribuir valores. Não existe maneira de se ler diretamente esses registradores.

Entretanto, o computador mantém uma cópia desses registradores na área de trabalho do sistema. Eles são armazenados na localização de memória entre RG0SAV e RG7SAV (veja a parte sobre a RAM do sistema na seção de referência sobre BIOS, para maiores detalhes).

VDP(8) é o registrador de leitura. Você pode ler esse registrador conforme é mostrado a seguir:

```
PRINT VDP(8)
```

Você precisa se lembrar que, se atribuir um número errado a esses registradores, o modo de tela pode ser interrompido. Assegure-se de saber o que está fazendo antes de usar a função VDP.

REGISTRADORES DO VDP

Registradores de Gravação

Registrador 0...VDP(0)

O registrador 0 contém dois bits de controle do VDP. BIT 1 (M3) e BIT 0 (EV). Os bits de 2 a 7 devem ser zero.

BIT 0 EV: VDP externo. Ele ativa/desativa a entrada de um VDP externo. EV = 0 para a maioria dos MSX.
0 = desativa
1 = ativa

BIT 1 M3: Modo de bit 3. M3 é usado para selecionar o modo de tela e é usado em conjunto com o registrador 1 (veja registrador 1).

RO

B7	B6	B5	B4	B3	B2	B1	B0
0	0	0	0	0	0	M3	EV=0

Registrador 1...VDP(1)

O registrador 1 contém sete bits de controle do VDP. O bit 2 é reservado e deve ser sempre zero.

BIT 0 MAG: Opção de ampliação de sprites.
0 = sem ampliação; resolução de 1 × 1 ponto.
1 = ampliado; resolução de 2 × 2 pontos.

BIT 1 SIZE: Seletor de tamanho para sprites.
0 = sprites de 8 × 8 pontos.
1 = sprites de 16 × 16 pontos.

BIT 2 Reservado = 0

BIT 3 M2: Bit de modo 2. M2 é usado para selecionar o modo de tela (veja o bit 4).

BIT 4 M1: Bit de modo 1. M1 é usado para selecionar o modo de tela. A combinação de M1, M2 e M3 dá o modo de tela.

M1	M2	M3	
0	0	0	Modo 1 de texto
0	0	0	Modo 2 de alta resolução gráfica
0	1	0	Modo 3 de gráfico multicolorido
1	0	0	Modo 0 de texto

BIT 5 IE: Ativador de interrupção
0 = desativa as interrupções do VDP.
1 = ativa as interrupções do VDP.

BIT 6 BL: Ativa/desativa a tela. Isto permite a você desativar a exibição da tela.
0 = altera a exibição da tela para branco e mostra somente a cor da margem.
1 = ativa a exibição da tela.

BIT 7 VRAM: Seleciona o tipo de RAM de vídeo usada. (VRAM=1 para a maioria dos MSX)
0 = 4K RAM
1 = 16K RAM

R1

B7	B6	B5	B4	B3	B2	B1	B0
VRAM	BL	IE	M1	M2	0	SIZE	MAG

Registrador 2...VDP(2)

O registrador 2 define o endereço base para a tabela de nomes. A sua faixa vai de 0 a 15. Esse registrador representa os 4 bits mais significativos dos 14 bits de endereço da tabela de nomes. Dessa forma, você deve escrever no endereço dividido por &H400.

R2

B7	B6	B5	B4	B3	B2	B1	B0
0	0	0	0	ENDEREÇO DA TABELA DE NOMES			

Nota: R2*&H400 = ENDEREÇO DA TABELA DE NOMES

Registrador 3...VDP(3)

O registrador 3 define o endereço base da tabela de cores. A sua faixa vai de 0 a 255. Este registrador representa os 8 bits maiores dos 14 bits do endereço da tabela de cores. Dessa forma, você deve escrever no endereço dividido por &H40.

Nota: R3*&H40 = ENDEREÇO DA TABELA DE CORES

Registrador 4...VDP(4)

O registrador 4 define o endereço base da tabela padrão, texto ou multicolorida, dependendo do modo de tela. A sua faixa vai de 0 a 7. Este registrador representa os 3 bits maiores dos 14 bits de endereço da tabela padrão, texto ou multicolorida. Dessa forma, você deve escrever no endereço dividido por &H800.

R4

B7	B6	B5	B4	B3	B2	B1	B0
0	0	0	0	0	TABELA DE ENDEREÇOS P.T.M.		

Nota: R4*&H800 = ENDEREÇO DA TABELA DE GERAÇÃO DE PADRÕES/ TEXTO/MULTICORES.

Registrador 5...VDP(5)

O registrador 5 define o endereço base da tabela de atributos de sprites. A sua faixa vai de 0 a 127. Este registrador representa os 7 bits mais significativos dos 14 bits de endereço da tabela de atributos de sprites. Dessa forma, você deve escrever no endereço dividido por &H80.

R5

B7	B6	B5	B4	B3	B2	B1	B0
0	ENDEREÇO DA TABELA DE ATRIBUTOS DOS *SPRITES*						

Nota: R5*&H80 = ENDEREÇO DA TABELA DE ATRIBUTOS DE *SPRITES*.

Registrador 6...VDP(6)

O registrador 6 define o endereço base da tabela geradora de padrões de sprites. A sua faixa vai de 0 a 127. Este registrador representa os 3 bits mais significativos dos 14 bits de endereço da tabela geradora de padrões de sprites. Dessa forma, você deve escrever no endereço dividido por &H800.

R6

B7	B6	B5	B4	B3	B2	B1	B0
0	0	0	0	0	END. TAB. GER PAD. *SPRITES*		

Nota: R6*&H800 = ENDEREÇO DA TABELA GERADORA DE PADRÕES DE *SPRITES*.

Registrador 7...VDP(7)

O registrador 7 é dividido em duas partes. Os 4 bits mais significativos contêm a cor do texto do modo texto, enquanto que os 4 bits menos significativos contêm a cor de fundo para os modos texto e gráfico.

A sua faixa vai de 0 a 15 para cada cor.

R7 = <cor de texto>*&H10+<cor de fundo>

R7

B7	B6	B5	B4	B3	B2	B1	B0
COR DE PRIMEIRO PLANO				COR DE FUNDO			

Registrador de Leitura

O Registrador de Status...VDP(8)

O Registrador de Status contém o indicador de pendência de interrupção, o indicador de colisão de sprites e o indicador do quinto sprite.

Indicador de Interrupção F: BIT 7

O indicador de interrupção está em 1 no final da análise do modo de tela. Ele é redefinido para 0 após o registrador de status ter sido lido, ou quando o VDP tiver sido redefinido externamente.

Indicador de Colisão de Sprites C: BIT 5

O VDP verifica se dois ou mais sprites se sobrepõem em um ponto. Isto é feito a cada 1/60 de segundo.

1 = colisão de sprite
0 = sem sobreposição

Indicador do Quinto Sprite 5S: BIT 6

5S é definido como 1 quando existir mais de quatro sprites em uma linha horizontal (linhas 0 a 192). Observe que o VDP não pode trabalhar com mais de 4 sprites em uma linha. Se existirem 5 sprites ou mais, o quinto sprite não será exibido.

Número do Quinto Sprite

Se existirem mais de 4 sprites em uma única linha horizontal e 5S estiver em 1, então o número de sprite do quinto sprite é fornecido nos cinco bits menos significativos.

R2

B7	B6	B5	B4	B3	B2	B1	B0
F	5S	C	NÚMERO DO QUINTO *SPRITE*				

CAPÍTULO 45

GRÁFICOS AVANÇADOS VI

A RAM DE VÍDEO

O MSX tem 16K de RAM de Vídeo (VRAM) que está separada da memória principal de programação. Ela é gerenciada e renovada pelo Processador de Vídeo (VDP) e, dessa forma, ela é independente do processador central Z80.

Isso fornece as seguintes vantagens:

1. O processador Z80 não precisa gerenciar a memória RAM do modo de tela, economizando tempo dessa forma.
2. Os gráficos e as exibições não usam área preciosa da memória RAM principal que é gerenciada pelo Z80.

A RAM de Vídeo é dividida em várias seções, cada uma com uma função única.

É possível acessar a RAM do Vídeo a partir do BASIC usando os seguintes comandos e funções:

BSE	Fornece o endereço base de várias tabelas na RAM do Vídeo.
VPEEK	Lê o conteúdo de um endereço da RAM de Vídeo.
VPOKE	Grava em um endereço da RAM de Vídeo.

A função BASE fornece a posição do endereço base atual de várias tabelas de modos de tela na RAM de vídeo (veja BASE para maiores detalhes).

Nome de Tabela e Endereço em Hexadecimal	**Modo texto 0**	**Modo texto 1**	**Modo gráfico 2**	**Modo gráfico 3**
de modelos	&H0000	&H1800	&H1800	&H0800
de cores		&H2000	&H2000	
de padrões dos caracteres	&H0800	&H0000	&H0000	&H0000
de atributos dos *sprites*		&H1B00	&H1B00	&H1B00
de padrões dos *sprites*		&H3800	&H3800	&H3800

CAPÍTULO 46

EFEITOS AVANÇADOS DE SOM USANDO O PSG

O processador gerador de som do MSX, o AY-3-8912, já existe há algum tempo. Ele foi desenvolvido pela General Instruments nos anos 70 e ainda é o mais popular processador de som. É muito usado em pequenos sistemas de computadores de 8 bits e máquinas de fliperama. As razões de seu sucesso são as suas habilidades para gerar até 3 canais de sons periódicos e um canal com ruído, simultaneamente. Além disso, por ser ele próprio um microprocessador, quando o som é gerado, o processador não pára as operações do processador central. Existem muitas coisas que você pode fazer com esse gerador de som.

Este capítulo trata sobre o gerador de som AY-3-8910 e o uso do comando SOUND.

Como exemplo, tente o pequeno programa a seguir, que gera um barulho fantástico de disco voador.

```
10 SOUND 0,87
20 SOUND 7,62
30 SOUND 8,16
40 SOUND 11,179
50 SOUND 12,45
60 SOUND 13,14
```

FORMAS DE ONDAS GERADAS PELO PSG

Basicamente, o PSG pode gerar apenas dois tipos de formas de ondas: uma onda periódica quadrada e ruído branco.

```
10 SOUND 7,62
20 SOUND 8,15
30 SOUND 1,1
40 SOUND 0,28
```

```
10 SOUND 7,55
20 SOUND 8,15
```

Entretanto, existem ondas de som que podem ser modeladas e modificadas em ruídos complexos de modo que tenham vida própria. Para fazer isso, você precisa controlar a freqüência, o volume, o tom e o timbre de cada canal de som usando o comando SOUND que acessa o PSG.

REGISTRADOR DE SOM DO PSG

O AY-3-8910 tem 14 registradores que você pode alterar usando o comando SOUND, que possui a seguinte sintaxe:

SOUND <número do registrador>,<valor a gravar>

Colocando valores adequados nesses registradores você pode selecionar canais, ondas de som, freqüências do tom, ruído e volumes.

REGISTRADOR	DESCRIÇÃO
0	Freqüência do canal A (menos significativo).
1	Freqüência do canal A (mais significativo).
2	Freqüência do canal B (menos significativo).
3	Freqüência do canal B (mais significativo).
4	Freqüência do canal C (menos significativo).
5	Freqüência do canal C (mais significativo).
6	Freqüência do gerador de ruído.
7	Ativa ou desativa os canais A, B e C para sons.

8	Controle de volume para o canal A.
9	Controle de volume para o canal B.
10	Controle de volume para o canal C.
11	Período do envelope (byte menos significativo).
12	Período do envelope (byte mais significativo).
13	Padrão do envelope.

ESCOLHENDO O SOM PARA CADA CANAL: REGISTRADOR 7, O MISTURADOR

Este é o registrador que ativa ou desativa o som para os três canais. Ele pode selecionar a fonte da onda sonora a partir do gerador de ruído ou a partir de um gerador individual de tom para cada canal.

Os bits 5, 4 e 3 do registrador 7 ativam o gerador de ruído para os canais C, B e A respectivamente. 1 significa desligado e 0 ligado. Os bits 2, 1 e 0 do registrador 7 ativam o gerador de tons para os canais C, B e A respectivamente. Dessa forma, por exemplo, se você quiser que o canal C emita um tom, o bit 2 deve estar posicionado em 0.

BIT	B7	B6	B5	B4	B3	B2	B1	B0
	/	/	RUÍDO			TOM		
	/	/	C	B	A	C	B	A
	0	0	1	1	1	0	1	1
			OFF	OFF	OFF	ON	OFF	OFF

O formato do comando SOUND para esta tabela é:

```
SOUND 7,&B00111011, ou, em decimal, SOUND 7,59
```

O tom do canal A depende dos registradores 0 e 1; daremos maiores informações sobre isso mais tarde.

Se você quiser que o canal B emita ruídos brancos, e os canais C e A conforme acima, então o registrador 7 se altera para:

BIT	B7	B6	B5	B4	B3	B2	B1	B0
	/	/	RUÍDO			TOM		
	/	/	C	B	A	C	B	A
	0	0	1	0	1	0	1	1
			OFF	ON	OFF	ON	OFF	OFF

O formato do comando SOUND para esta tabela é:

SOUND 7,&B00101011, ou em decimal, SOUND 7,43

Você pode fazer um canal emitir ruídos e tons se você ativar ambos. Digamos que queira que todos os canais emitam tanto tons como ruídos; você teria:

BIT	B7	B6	B5	B4	B3	B2	B1	B0
	/	/	RUÍDO			TOM		
	/	/	C	B	A	C	B	A
	0	0	0	0	0	0	0	0
			ON	ON	ON	ON	ON	ON

O formato do comando SOUND para esta tabela é:

SOUND 7,&B00000000, ou, em decimal, SOUND 7,0

Nota: Os bits B7 e B6 não são usados, e por esse motivo devem ser definidos como 0.

GERADOR DE TONS: OS REGISTRADORES 0, 1, 2, 3, 4, 5

Para definir a freqüência de um gerador de tom para cada canal, você deve alterar os registradores 0, 1, 2, 3, 3, 4 e 5. Cada canal tem dois registradores para fixar a freqüência do tom a ser gerado.

Registradores 0 e 1 para o canal A.
Registradores 2 e 3 para o canal B.
Registradores 4 e 5 para o canal C.

REGISTRO	FAIXA	DESCRIÇÃO
0	0-255	Ajuste fino da freqüência do canal A.
1	0-15	Ajuste básico da freqüência do canal A.
2	0-255	Ajuste fino da freqüência do canal B.
3	0-15	Ajuste básico da freqüência do canal B.
4	0-255	Ajuste fino da freqüência do canal C.
5	0-15	Ajuste básico da freqüência do canal C.
6	0-31	Freqüência do gerador de som.

O intervalo de freqüência (f) é calculado conforme a seguir:

f = (Registrador 0) + (Registrador 1) × 256

Esta fórmula não fornece a freqüencia em Hz, mas quanto maior a freqüência menor é o conteúdo do registrador.

Dessa forma, a freqüência mais alta é gerada por SOUND 0,0: SOUND 1,0 que fornece $f = 0$; e a freqüência mais baixa é fornecida por SOUND 0,255: SOUND 1,15.

Para calcular a freqüência gerada em Hz, use a seguinte fórmula:

$$f = \frac{178900}{(16 \times Hz)}$$

178900 é a freqüência do relógio do PSG (podendo variar de modelo para modelo).

Se você quiser ser exato sobre que freqüência quer que o PSG gere, deve fazer o cálculo anterior. Não! Não com uma calculadora; em vez disso use o computador:

```
10 INPUT "FREQUENCIA EM HERTZ";HZ
20 F=INT(1789800#/(16*HZ))
30 PRINT"AJUSTE FINO,REGISTROS 0,2,ou 4: ";F MOD 256
40 PRINT"AJUSTE BASICO,REGISTROS 1,3,ou 5: ";INT(F/256
)
```

O programa anterior faz o cálculo.

Exemplo:

Gere uma onda sonora de 2.000 Hz no canal A.

f=178900/(16·2000)=55,9

```
10 SOUND 0,255
20 SOUND 1,0
30 SOUND 7,62
40 SOUND 8,10
```

**FREQÜÊNCIA DO GERADOR DE RUÍDO:
O REGISTRADOR 6**

Para alterar a freqüência do gerador de ruído branco, você deve alterar o registrador 6 do PSG.

A faixa varia de 0 a 31; quanto menor o valor, maior a freqüência.

Exemplo:

```
10 SOUND 7,55
20 SOUND 8,15
30 FOR F=0 TO 31
40 SOUND 6,F
50 FOR I=1 TO 200: NEXT
60 NEXT F
```

Você deve ouvir uma queda gradual na freqüência dos ruídos brancos gerados.

VOLUME: REGISTRADORES 8, 9, 10

Os volumes dos sons emitidos pelos canais A, B e C são controlados pelos registradores 8, 9 e 10 respectivamente.

```
Volume mínimo              Volume máximo
0<------------------------------>15
REGISTRADOR  8 = CANAL A
REGISTRADOR  9 = CANAL B
REGISTRADOR 10 = CANAL C
```

Quando você coloca o valor 16 nesses registradores, o timbre é alterado. Isto significa que o volume se altera de acordo com a modulação do timbre e da freqüência. O timbre é determinado pelos registradores 11, 12 e 13.

Você deve se lembrar de que o volume de saída de uma TV depende do controle de volume. Se o controle de volume estiver desligado, você não ouvirá nada. O volume do som da saída de áudio do MSX também depende de um amplificador externo. No caso do Expert existe um alto-falante interno que permite trabalhar com geração de sons até com o volume da TV no mínimo.

COMO USAR TIMBRES: REGISTRADORES 11, 12 E 13

Quando não se usa timbre, o volume do som gerado permanece constante em um nível fixado pelos registadores 8, 9 e 10. Isso é um tanto monótono, pois tudo o que você ouve é um som de nível constante. Entretanto, o PSG AY-3-8910 tem uma facilidade de controle de timbre que pode alterar o volume periodicamente, de acordo com um dos modos de timbre disponíveis.

O timbre é ativado posicionando-se o controle de volume dos registradores 8, 9 e 10 em 16. Quando o timbre é ativado, o volume do som se altera de acordo com os registradores 11, 12 e 13.

TIMBRE DESATIVADO		TIMBRE ATIVADO
CANAL A, REGISTRADOR 8	0–15	16
CANAL B, REGISTRADOR 8	0–15	16
CANAL C, REGISTRADOR 8	0–15	16

A modulação do padrão de timbre pode ser selecionada alterando-se o o registrador 13 para um número entre 0 e 15. Entretanto, o PSG duplica alguns dos padrões, de modo que você consegue somente 8 timbres diferentes.

A taxa de mudança de volume ou período do timbre pode ser alterada usando-se os registradores 11 e 12; o registrador 12 fornece uma alteração de período de graves e o registrador 11, uma alteração precisa do período.

	FAIXA
registrador 11	0–255
registrador 12	0–255

Período = (registrador 12) + 256*(registrador 11)

Observe que você só pode ter um timbre por vez, pois todos os três canais usam o mesmo.

Veja SOUND na seção de referência BASIC para mais detalhes sobre a modulação de timbre.

CAPÍTULO 47

COMO USAR ARQUIVOS

No MSX-BASIC é possível gravar dados, bem como o conteúdo de uma matriz string para o cassete ou para o disquete, e de forma inversa lê-los de lá. Esse conjunto de dados é chamado "arquivo". Existe um conjunto de comandos e funções do BASIC que permitem que você grave e leia em uma fita cassete ou em um disquete.

MAXFILES

Antes de usar um arquivo é aconselhável aumentar o parâmetro MAXFILES, uma função usada para definir o número máximo de arquivos abertos ao mesmo tempo. O valor padrão de MAXFILES é 1 mas não é normalmente suficiente. O número máximo para MAXFILES é 15.

Falando tecnicamente, MAXFILES tem o efeito de aumentar o tamanho do Bloco de Controle de Arquivos na memória. O Bloco de Controle de Arquivos é usado como área de trabalho pela ROM do MSX-BASIC quando você estiver usando arquivos (veja o mapa de memória, Capítulo 20).

OPEN

Para dizer ao computador que está começando a ler ou gravar arquivos, você

deve usar o comando OPEN para informar que um dispositivo específico está aberto. Ele aloca uma área de memória auxiliar no Bloco de Controle de Arquivos e define o modo de operação de entrada e saída para essa memória.

Você precisa executar OPEN antes dos seguintes comandos:

PRINT#	PRINT# USING
INPUT#	LINE INPUT#
INPUT$(#)	EOF

Sintaxe

OPEN "<nome do dispositivo>[<nome do arquivo>]"[FOR<modo>] AS#<número do arquivo>

Existem cinco dispositivos básicos na atual versão do MSX. Eles são:

CAS:	Cassete
CRT:	Modo de tela de texto
GRP:	Modo gráfico de tela
LPT:	Impressora de linha
A: ou B:	Unidade de disco flexível (disquete)

Dos dispositivos acima, somente CAS:, GRP: e A: (ou B:) têm uso real. CAS: é especificado quando você está usando um cassete para armazenar ou recuperar dados; GRP: é especificado quando você está tentando gravar num modo gráfico de tela; e A: ou B: quando você usa disquetes. Não é possível imprimir texto nos modos de tela 2 e 3, de forma que você tem de enviar informações para um modo gráfico como se ela fosse um arquivo. Uma descrição detalhada de como usar arquivos para imprimir num modo gráfico de tela é dada no Capítulo 14.

Existem três modos de entrada e saída:

OUTPUT	Especifica modo de saída seqüencial, isto é, grava num dispositivo usando PRINT# E PRINT# USING.
INPUT	Especifica modo de entrada seqüencial, isto é, lê de um dispositivo usando INPUT# etc.
APPEND	Especifica modo seqüencial append.

<número de arquivo> é um inteiro cujo valor está entre um e MAXFILES, o número máximo de arquivos. Os números de arquivos são informados de forma que, quando colocados entre aspas nos comandos PRINT#, por exemplo, o computador sabe em que dispositivo deve ler ou gravar. O <número de arquivo> é associado com o arquivo quando este é aberto. Ele deve ser um dos que já estejam sendo usados no programa.

Exemplo:

Vamos dizer que queremos abrir um arquivo de nome INFO em um cassete para gravar informações. O comando OPEN será usado assim:

```
OPEN "CAS:INFO" FOR OUTPUT AS# 1
```

CLOSE

A função oposta a OPEN é executada pelo comando CLOSE. É conveniente fechar um arquivo assim que você terminar de usá-lo.

PRINT#

Para gravar em vários dispositivos, usa-se um comando especial de impressão, o PRINT#. O sinal "#" é seguido pelo número do arquivo especificado no comando OPEN.

Quando usado com o cassete como dispositivo, PRINT# tem o efeito de gravar as informações fornecidas na fita cassete.

PRINT# USING

Esta é a versão para arquivo do comando PRINT USING e tem o mesmo efeito. Seu uso principal é imprimir no modo gráfico de tela e num formato especificado.

INPUT#

Para ler dados de dispositivos diferentes do teclado como um disquete, nós usamos o comando INPUT#. Ele tem o mesmo efeito que o comando INPUT, mas em vez

de se digitar a informação requerida a partir do teclado, ela é lida a partir do dispositivo especificado pelo comando OPEN; na maioria dos casos, esse dispositivo é o cassete ou o disquete.

INPUT(#)
LINE INPUT(#)

Essas são as versões para arquivos dos comandos INPUT$ e LINE INPUT, respectivamente (veja INPUT#).

EOF

EOF produz um retorno quando se tiver chegado ao final do arquivo. Se o final do arquivo for detectado, EOF retorna –1; de outra forma, é sempre 0.

Programa-exemplo

Eis um programa de demonstração que deve dar a você uma idéia de como usar arquivos.

O programa requer que você digite cinco nomes e os armazene em um arquivo no cassete. Então ele pergunta se você quer recuperar os nomes.

```
10 DIM N$(5)
20 MAXFILES=4
30 FOR I=1 TO 5
40 INPUT"POR FAVOR DIGITE UM NOME";N$
50 NEXT I
60 PRINT"O GRAVADOR ESTA LIGADO? (S=OK)"
70 IF INKEY$<>"S"AND INKEY$<>"s" THEN 70
80 OPEN "CAS:" FOR OUTPUT AS #2
90 FOR I=1 TO 5
100 PRINT#2,N$(I)
110 NEXT I
120 CLOSE #2
130 PRINT"PRONTO PARA CARREGAR?(S=OK)"
140 IF INKEY$<>"S"AND INKEY$<>"s" THEN 140
150 OPEN "CAS:" FOR INPUT AS #3
```

```
160 FOR I=1 TO 5
170 INPUT#3,N$(I)
180 NEXT I
190 CLOSE#3
200 FOR I=1 TO 5
210 PRINT N$(I)
220 NEXT I
```

LINHA 10	MATRIZ PARA OS NOMES
LINHA 20	DEFINE O NÚMERO MÁXIMO DE ARQUIVOS COMO 5
LINHA 30	RECEBE CINCO NOMES DE MODO QUE POSSAMOS FAZER ALGUMA COISA COM ELES
LINHA 60	ATIVA O CASSETE (PARA GRAVAR)
LINHA 70	ESPERA QUE O CASSETE SEJA LIGADO
LINHA 80	ABRE O ARQUIVO PARA O CASSETE
LINNA 90	JOGA A SAÍDA PARA O CASSETE USANDO PRINT# USING
LINHA 120	ENTÃO FECHA O ARQUIVO QUANDO TERMINA
LINHA 130	REBOBINE O CASSETE DE MODO QUE POSSAMOS CARREGAR O ARQUIVO GRAVADO
LINHA 150	ABRE O ARQUIVO PARA ENTRADA
LINHA 160	LÊ O ARQUIVO DO CASSETE USANDO INPUT#
LINHA 190	FECHA O ARQUIVO QUANDO TERMINA
LINHA 200	IMPRIME OS RESULTADOS

```
RUN
POR FAVOR DIGITE UM NOME? OS
POR FAVOR DIGITE UM NOME? DANI
POR FAVOR DIGITE UM NOME? RA
POR FAVOR DIGITE UM NOME? FONKS
POR FAVOR DIGITE UM NOME? GUIZIN
O GRAVADOR ESTA LIGADO? (S=OK)
PRONTO PARA CARREGAR?(S=OK)
OS
DANI
RA
FONKS
GUIZIN
```

CAPÍTULO **48**

MAPA DA MEMÓRIA

Endereço	Área	
&HFFFF	ÁREA DE TRABALHO DO SISTEMA	
&HF380	BLOCO DE CONTROLE DE ARQUIVOS	
	ÁREA DE STRINGS	
	PILHA	
	ÁREA LIVRE	
	ÁREA DE MATRIZES	
	VARIÁVEIS	
	ÁREA DO PROGRAMA BASIC	
&H8000	ROM DO BASIC MSX-BASIC	&HC000 para MSX com 16K
&H0000		

ÁREA DE TRABALHO

A área de trabalho está situada entre as localizações de memória &HFFFF e &HF380 e é usada pela ROM do MSX-BASIC para suas operações internas. Esta área contém as diversas variáveis do sistema, sendo que uma lista das mesmas é fornecida na seção de referência da BIOS.

BLOCO DE CONTROLE DE ARQUIVOS

Esta área é reservada para operações de entrada e saída usando arquivos. Comandos como PRINT# e INPUT# usam este bloco.

O tamanho do bloco de controle de arquivos é alterado usando-se a função MAXFILES.

O limite superior do bloco de controle de arquivos é &HF380. Entretanto, você pode diminuir esse limite para criar uma área de memória para armazenar programas em código de máquina e dados. Isto é feito usando-se o comando CLEAR que reduz esse limite.

Exemplo: Criar uma área de usuário para linguagem de máquina entre &HF380 e &HF000:

```
CLEAR ,&HF000
```

Endereço	Área
&HFFFF	ÁREA DE TRABALHO DO SISTEMA
&HF380	ÁREA DE CÓDIGO DE MÁQUINA DO USUÁRIO
&HF000	BLOCO DE CONTROLE DE ARQUIVOS

ÁREA DE STRINGS

Esta área armazena o conteúdo das variáveis string. O tamanho padrão desta área é de 200 bytes. Isso define o limite máximo reservado para armazenar strings em 200 caracteres. Entretanto, o espaço pode ser aumentado usando-se o comando CLEAR.

Exemplo: Aumentar a área de strings para 255 bytes.

```
CLEAR 255
```

ÁREA DE ARMAZENAMENTO

Ela armazena os "endereços" para loops FOR/NEXT e comandos GOSUB. Ela armazena os endereços de onde o BASIC deve retornar em um comando NEXT ou RETURN.

ÁREA LIVRE

Esta é uma área não usada e disponível para o usuário. O seu tamanho pode ser visto usando o comando FREE conforme a seguir:

```
PRINT FREE(0)
28815          Para sistemas MSX com 32K ou 64K, ao se ligar o computador.
12431          Para sistemas MSX com 12K, ao se ligar o computador.
```

Observe que o tamanho dessa área diminui à medida que você aumenta o tamanho do programa BASIC ou das variáveis.

ÁREA DIM

Esta área armazena as variáveis de matrizes DIM e é aumentada durante a execução de um comando DIM. Se uma matriz for uma matriz string, os ponteiros (pointers) para a área de string serão armazenados na área DIM.

ÁREA DE VARIÁVEIS

Esta área armazena variáveis numéricas, e um ponteiro descritor de string para a área de strings também será armazenado aí.

O endereço de armazenamento de uma variável em particular pode ser visto usando-se o comando VARPTR (veja VARPTR na Parte 3, Volume 2).

CAPÍTULO 49

O COMANDO USR E A LINGUAGEM DE MÁQUINA

Para executar sub-rotinas em linguagem de máquina a partir do BASIC, use o comando USR. O comando USR chama um programa em linguagem de máquina para a localização de memória especificada no comando DEF USR. Você pode usar até 10 rotinas em linguagem de máquina simultaneamente. Certamente, é possível ter mais de de dez rotinas, redefinindo-se o comando USR.

COMO DEFINIR O COMANDO USR

Para chamar uma rotina em linguagem de máquina, o computador deve saber o endereço inicial da rotina. Use o comando DEF USR para atribuir um endereço a USR.

Exemplo: Digamos que uma rotina em linguagem de máquina se inicia em &HF000:

```
DEF USR0=&HF000     (você pode eliminar o 0 de modo que DEF USR = &HF000)
```

COMO EXECUTAR UMA ROTINA EM LINGUAGEM DE MÁQUINA

O comando USR pode ser usado da mesma maneira que um comando comum. Isto significa que ele pode estar em uma expressão, como abaixo:

```
X = USR(99)
YS = "M/C"=USR9("abx")
PRINT USR7(0)
```

O exemplo executa a sub-rotina e retorna um valor dependendo do que o programa em linguagem de máquina faz. Os números ou strings entre parênteses são parâmetros a serem passados para o programa em linguagem de máquina. Por causa disso a maneira como você usa o comando USR realmente depende da sua rotina em linguagem de máquina. Você pode ter um argumento nela, ou somente um parâmetro fictício; ela também pode retornar um parâmetro ou nada.

Se você quiser executar um programa em linguagem de máquina sem parâmetros de entrada ou de saída, então use uma variável fictícia:

```
X = USR(0)
```

onde X é uma variável fictícia e (0) é um parâmetro fictício.

COMO PASSAR UM PARÂMETRO PARA UMA ROTINA EM LINGUAGEM DE MÁQUINA A PARTIR DO BASIC

Você pode passar qualquer tipo de parâmetro para o programa em linguagem de máquina a partir do BASIC usando o comando USR<número>(<parâmetro>). O parâmetro deve estar entre parênteses. Quando o MSX executa o comando, ele verifica o tipo de argumento usado, isto é, se o parâmetro é um inteiro, um número de precisão simples ou uma string.

Se existir um parâmetro, o programa deve informar o tipo e a localização na memória. O MSX-BASIC classifica o tipo de parâmetro antes que o programa em linguagem de máquina seja executado.

Para descobrir o tipo de parâmetro que foi passado, examine o endereço de memória &HF663:

&HF663 = 2 = 00000010	parâmetro numérico inteiro
&HF663 = 4 = 00000100	parâmetro numérico de precisão simples
&HF663 = 8 = 00001000	parâmetro numérico de precisão dupla
&HF663 = 3 = 00000011	string

Localização do parâmetro passado.

Inteiro (&HF663=2) – de &HF7F8 a &HF7F9

&HF7F8 = <byte menor>

&HF7F9 = <byte maior>

<parâmetro inteiro>=<byte menor> +256* <byte maior>

Número de Precisão Simples (&HF663=4) – de &HF7F6 a &HF7F9

7HF7F6 =] valor
7HF7F7 =] armazenado em decimal
7HF7F8 =] codificado em binário
7HF7F9 =

Número de Precisão Dupla (&HF663=8) – de &HF7F6 a &HF7FD

7HF7F6 =] valor do parâmetro
7HF7F7 =] armazenado em decimal
7HF7F8 =] codificado em binário desde a posição
7HF7F9 =] &HF7F6
7HF7FA =]
7HF7FB =]
7HF7FC =]
7HF7FD =]

String (&HF663=3) – de &HF7F8 a &HF7F9

7HF7F8 =]
7HF7F9 =] ... <endereço 1> endereço do bloco de descrição de string

<endereço 1> = tamanho da string
<endereço 1> + 1 = byte menos representativo do endereço
<endereço 1> + 2 = byte mais representativo... <endereço 2>, localização da string

COMO RETORNAR UM PARÂMETRO A PARTIR DA LINGUAGEM DE MÁQUINA

Para retornar um parâmetro para o MSX-BASIC, defina o endereço &HF663 com o valor apropriado de acordo com o tipo de dado e armazene os seus parâmetros nos endereços em questão antes de sair do programa em linguagem de máquina. O BASIC pega o que existir no parâmetro armazenado e transfere-o para uma variável após ter deixado a rotina (programa) em linguagem de máquina.

CAPÍTULO **50**

GERENCIAMENTO DE MEMÓRIA DO MSX E O MECANISMO DOS CARTUCHOS

INTRODUÇÃO

A CPU do MSX e o microprocessador Z80A, tem um *bus* de endereçamento de 8 bits que pode endereçar até 64Kbytes de memória a qualquer tempo. Os 64Kbytes de memória são divididos em 4 "páginas" de 16K de memória. O MSX pode expandir essa memória mudando para diferentes "slots" para cada página. Um "slot" é basicamente um espaço de memória que pode ser entendido como um banco de memória separado.

Fisicamente, um slot pode ser um slot de cartucho ou um banco físico de memória como a ROM do MSX-BASIC ou o seu sistema RAM.

Para selecionar o slot a ser usado com cada página, o registrador de seleção de slot tem de ser definido. Veremos detalhes posteriormente.

Normalmente, um MSX comum tem a seguinte configuração:

PÁGINA	SEGMENTO DO SISTEMA	SEGMENTOS 1, 2, 6, 3 CARTUCHO A CONECTAR
Página 3	RAM de 16K para os MSX de 16K	
Página 2	RAM de 16K para os MSX de 32K	ROM de 8 ou 16K para programas de jogos Extensão de 16K para a RAM de MSX de 16K.
Página 1	ROM do BASIC do MSX	Empregada para uma extensão do BASIC, ROM com o sistema operacional de disquete para outras linguagens
Página 0	ROM de BASIC do MSX (BIOS)	

CARTUCHO

Todos os computadores MSX têm pelo menos um slot de cartucho padrão de MSX, onde você pode conectar vários dispositivos. Eis uma lista dos possíveis dispositivos de cartucho:

1. Expansão de RAM. Normalmente para expandir uma máquina de 16K para 32K.

2. Cartucho ROM com programas de jogos. O cartucho contém ou código de máquina ou um programa em BASIC.

3. Cartucho ROM de expansão do BASIC. Esta ROM contém rotinas para expandir o MSX-BASIC. As rotinas são acessadas usando-se o comando CALL. Alguns cartuchos têm suas próprias áreas de trabalho em RAM.

4. Cartucho de dispositivo de entrada e saída. Esse dispositivo pode ser um controlador de disco flexível ou uma conexão com impressora ou um cartucho para caneta luminosa. Normalmente ele vem com o seu próprio utilitário em ROM para controlar a entrada e a saída.

Qualquer um dos dispositivos citados pode ser conectado a qualquer slot. O MSX tem um mecanismo de seleção de slots de modo que ele sabe que cartucho acessar.

ARRANJO BÁSICO DE SLOTS

O MSX tem quatro slots básicos, sendo o slot número 0 o que possui a ROM do MSX-BASIC e é chamado slot do sistema. Cada um desses quatro slots pode ser expandido para 4 slots de expansão. Dessa forma, o número total de slots expandíveis é 16. Esses 16 slots podem ter todos 64K de RAM dando a você 1Mbyte de espaço de memória. Isso é o máximo de RAM que o MSX pode tratar. Entretanto, você deve ter em mente que não se pode acessar toda a memória de 1Mbyte com o BASIC! Você precisa do MSX-DOS ou de programas em linguagem de máquina para fazer uso efetivo do espaço de memória acima de 64K.

Configuração básica de slots:

Configuração de slot expandido:

SELETOR DE SLOTS

O MSX pode ter diversos slots ou bancos de memória mas, a menos que eles sejam controlados de alguma forma, podem interferir uns com os outros. De modo a classificar que slot deve ser usado para que página, o MSX tem um mecanismo especial para selecionar slots.

A qualquer tempo, a CPU pode acessar os 64K ou as 4 páginas de RAM. Essas 4 páginas podem estar, cada uma, em slots diferentes. As páginas nos slots são selecionadas usando a porta A de saída de 8 bits do PPI 8255 (veja o circuito selecionador de slots na próxima página).

PA0 e PA1 fornecem o número de slot para a página 0.

PA2 e PA3 fornecem o número de slot para a página 1.

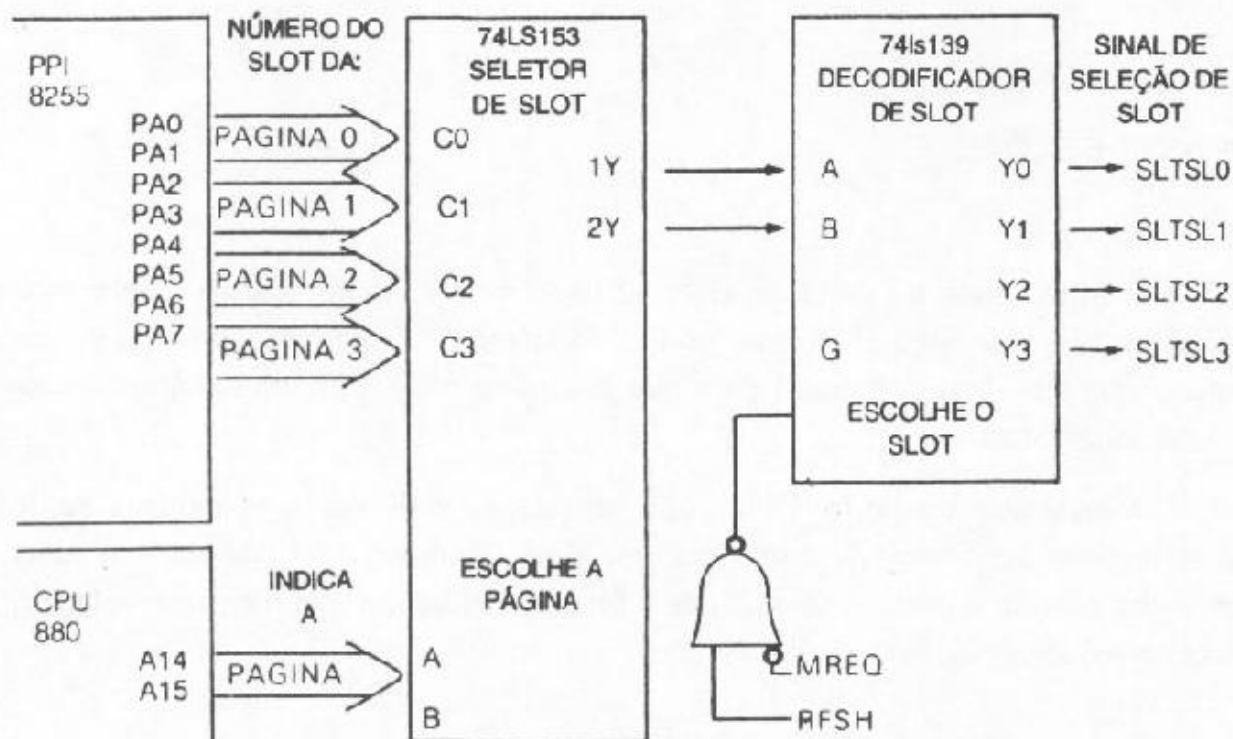
PA3 e PA4 fornecem o número de slot para a página 2.

PA5 e PA6 fornecem o número de slot para a página 3.

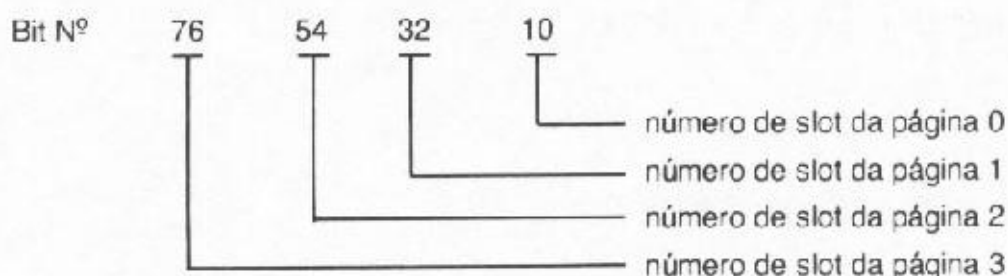
O sinal do PPI é enviado para o selecionador de slot 74LS153. Esse chip também recebe sinais da CPU Z80 para a página atual da qual a CPU está lendo e para a qual está gravando.

Sinal A15	Sinal A14:	O número da página que a CPU está lendo ou gravando
0	0	página 0
0	1	página 1
1	0	página 2
1	1	página 3

O número de slot selecionado é então passado no decodificador 2 para 4, 74LS139, que fornece o sinal de slot selecionado, SLTSL, para o slot relacionado das 4 páginas.

DIAGRAMA DO CIRCUITO SELECIONADOR DE SLOT

COMO SELECIONAR E ATIVAR UM SLOT

Para selecionar slots para cada página com um software, faça a saída para o endereço &HA8, que é a saída para a porta A do PPI.

O valor a ser saído é um número de 8 bits.

Exemplo:

Digamos que você quer usar o slot 0 para as páginas 0, 1 e 3 e o slot 1 para a página 2.

Bit Nº	7	6	5	4	3	2	1	0
	0	0	0	1	0	0	0	0 = 16

Você deve enviar 16 para a porta de entrada e saída &HA8. Entretanto, é melhor usar a chamada da BIOS ENASLT (&H0024) a partir do código de máquina.

EXPANSÃO DE SLOTS

O MSX tem a opção de até 4 slots básicos, de 0 a 3. Nem todos eles são fornecidos com todas as máquinas, mas cada slot pode ser expandido usando-se o kit de expansão. Dessa forma, o número máximo de slots é 16. Como todos eles podem receber até 64K de RAM, a memória RAM máxima que o MSX pode ter é 1Mbytes.

Esses slots de expansão não podem ser expandidos de qualquer maneira. Eles são sempre expandidos a partir de um slot básico, de modo que não tem sentido conectar um kit de expansão em um slot já expandido.

Para selecionar um slot de expansão, você primeiro deve selecionar o slot básico, usando a porta A do PPI 8255. O registro selecionador de slots está na localização &HFFFF no slot expandido. Isto determina se a página selecionada nesse slot está sendo usada ou não.

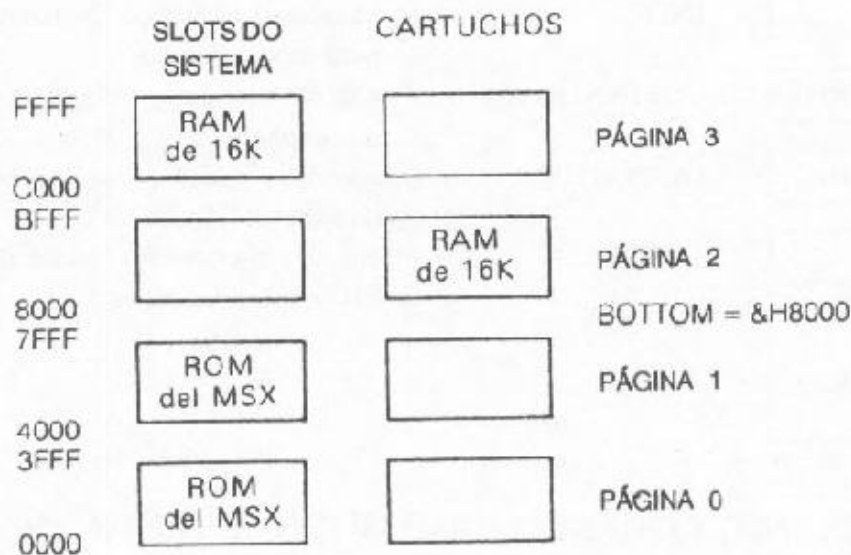
Para descobrir se um slot básico tem um kit de expansão conectado ou não, coloque no endereço &HFFFF o valor &H0F (POKE &HFFF, &H0F). Se você então ler o endereço &HFFFF (PRINT PEEK (&HFFFF)) e obtiver um complemento do que você gravou, isso significa que existe um kit de expansão aí.

PROCEDIMENTO DE PESQUISA DE RAM

Quando o computador MSX é ligado, ele pesquisa todos os slots para selecionar a RAM do sistema:

1. Ele primeiro pesquisa pela RAM disponível na página 2, de &H8000 a &HBFFF e então ativa o slot com a RAM maior da página 2. Se existir mais de um slot com o mesmo espaço de memória, então o slot com o menor número de slot é selecionado.

2. Logo, ele repete o mesmo processo para a página 3, de &HC000 a &HFFFF. Outra vez, a maior RAM disponível com o menor número de slot é selecionada.

3. Por último ele verifica se a RAM é contínua de &H8000 a &HFFFF e posiciona na variável BOTTOM do sistema (&HFC48) o endereço da menor RAM disponível.

CARTUCHOS DE ROM COM PROGRAMAS

Procedimento de Pesquisa da ROM

Tendo selecionado a RAM do sistema, o MSX então pesquisa os cartuchos de ROM de &H4000 a &HBFFF, página 1 e página 2. Ele pesquisa por um ID (identificador) válido no início de cada página a partir do slot 0 até o slot 3, e os slots de expansão quando expandidos.

A área de ID está localizada no início da página seguinte e pode ter um dos seguintes formatos:

Endereço	Campo		Descrição
+&H00	ID'AB'	ID ...	2 bytes código 'AB' para indicar que existe um cartucho de ROM aí.
+6H02	INIT	INIT ...	contém o endereço da rotina de inicialização para esse cartucho.
+&H04	COMANDO	COMANDO ...	contém o endereço do gerenciador de comandos expandido.
+&H06	DISPOSITIVO	DISPOSITIVO ...	contém o endereço do gerenciador de dispositivo expandido.
&H08	TEXTO	TEXTO ...	contém o endereço inicial do programa BASIC contido nesse cartucho.
&H0A &H10	RESERVADO		

Nota: ID, INIT, COMANDO, DISPOSITIVO e TEXTO irão conter zeros se não forem aplicáveis.

O MSX-BASIC toma as seguintes ações para o procedimento de pesquisa:

1. Verifica a área de ID e descobre que tipo de rotina tem. Ele então passa a informação coletada para a área de trabalho relacionada.
2. Executa a rotina INIT, se existir.
3. Executa o programa BASIC desse cartucho, se existir.

Nota: COMANDO e DISPOSITIVO não são executados nesse ponto porque eles somente são usados quando o usuário solicita um comando ou dispositivo de expansão.

INIT: Rotina de Inicialização

INIT contém o endereço da rotina de inicialização específica para o cartucho. Ela normalmente inicializa os dispositivos de entrada e saída conectados a esse cartucho ou posiciona uma conexão (hook) (veja mais a respeito na seção de *Conexão – Hook*), ou reserva uma área de trabalho para o software do cartucho.

A rotina de inicialização pode alterar todos os registros exceto o ponteiro de pilha (stack pointer). Ele retorna para o BASIC usando o comando do Z80 RET>

O INIT não precisa ser uma rotina de inicialização. Ele pode ser um programa em código de máquina a ser executado imediatamente após ligar o computador. Pode ser um jogo.

Se não existir rotina de inicialização, então INIT contém zeros.

TEXT: O Programa BASIC

Um cartucho de ROM não precisa ter necessariamente um programa em código de máquina: ele pode ser escrito em BASIC. TEXT contém o endereço inicial do programa BASIC contido nesse cartucho.

Quando se programa um software de cartucho em BASIC, as seguintes normas devem ser lembradas:

1. Quando existir mais de um cartucho com software BASIC conectado, a máquina irá executar somente aquele do slot com o menor número.
2. O programa BASIC do cartucho está armazenado no formato "tokens".
3. O cartucho deve ser colocado na página 2, &H8000 a &HBFFF. Isto significa que a memória máxima do BASIC é 16K.
4. Uma RAM que esteja colocada na página 2 de qualquer outro slot será desativada e dessa forma não poderá ser usada a partir do programa BASIC em cartucho.
5. O endereço apontado pela entrada TEXT deve conter um 0.
6. Os números de linhas para GOTO, GOSUB etc., devem ser alterados para ponteiros, para obter-se uma execução mais rápida.

Se não existir programa BASIC no cartucho, então TEXT contém zeros.

COMANDO (Statement): Rotina de comando expandido

Usando o comando CALL no BASIC do MSX, você pode usar comandos expandidos chamados de fora da ROM do BASIC do MSX. Um cartucho contendo um comando BASIC expandido deve conter o endereço inicial da primeira rotina de comando

expandido em STATEMENT. O cartucho deve ser colocado na página 1, mas pode ser qualquer slot, exceto o do sistema onde reside o BASIC.

A sintaxe para o comando expandido é:

CALL <nome do comando>
CALL <nome do comando> (argumentos)

CALL pode ser abreviado pelo caractere sublinhado '_'.

Quando o BASIC encontra um comando CALL, ele armazena o nome do comando na área de trabalho do sistema, PROCNM. PROCNM está localizado em &HFD89 e tem 16 bytes de tamanho. O nome do comando termina com um zero quando está armazenado em PROCNM, de modo que o tamanho máximo de um comando expandido é 15 caracteres.

Antes do comando expandido ser executado, o ponteiro de texto, o registro HL, aponta para o endereço após o nome do comando.

O procedimento de comando do cartucho então verifica o conteúdo de PROCNM e executa a rotina que coincida com o nome do comando.

Após ele ter executado o comando expandido, o flag de execução é limpo e o ponteiro de texto (HL) é apontado para a posição do comando seguinte.

Se não existir comando expandido que coincida com o nome em PROCNM, então o flag de execução e o ponteiro de texto permanecerão como estão, e retornarão para o BASIC com a mensagem "Syntax Error" (Erro de Sintaxe).

Se não existir rotina de comando expandido no cartucho, então STATEMENT contém zeros.

DEVICE: Rotina de Tratamento do Dispositivo de Expansão

O MSX-BASIC tem a habilidade de se conectar com uma expansão de dispositivo de entrada e saída para um slot de cartucho. DEVICE contém o endereço inicial da rotina de gerenciamento de dispositivo.

As seguintes notas devem ser lembradas acerca da rotina DEVICE.

1. O cartucho deve ser colocado na página 1, &H4000–&HBFFF.

2. Um cartucho (16K) pode ter até 4 dispositivos lógicos conectados.

3. O nome do dispositivo é armazenado na área de sistema PROCNM, da mesma forma que com STATEMENT. PROCNM está localizado de &HFD89 em diante e tem 16 bytes de comprimento. O nome do dispositivo termina com um zero quando armazenado em PROCNM, de modo que o tamanho máximo de um comando expandido é de 15 caracteres.

4. Quando o BASIC encontra um nome de dispositivo, em um comando OPEN ou em outros, que já conhecido pela ROM do MSX, então ele chama a entrada do dispositivo com &HFF no registrador A. Se não existir gerenciador que coincida com o nome do dispositivo, o flag de execução será posicionado. Se existir um dispositivo, ID (de 1 a 3) deve ser retornado no registrador A e o flag de execução será reposicionado. Todos os registradores podem ser alterados na rotina de dispositivo.

5. Uma operação real de entrada e saída é executada quando a entrada DEVICE é feita com um dos seguintes valores no registrador A:

0	Open
2	Close
4	Entrada e saída aleatória
6	Saída seqüencial
8	Entrada seqüencial
10	Função LOC
12	Função LOF
14	Função EOF
16	Função FPOS
18	Caractere de Cópia de Segurança

A variável do sistema, DEVICE, deve ser posicionada para número de Dispositivo ID (0–3).

Se não existir rotina de DEVICE no cartucho, então DEVICE deve conter zeros.

DESCRIÇÕES DAS VARIÁVEIS DO SISTEMA RELACIONADAS COM O MECANISMO DE SLOT

Status de cada Slot

EXPTBL – Indica qual slot está expandido.
Localização &HFCC1 – 4 bytes de comprimento.

EXPTBL &HFCC1 para o slot 0
&HFCC2 para o slot 1
&HFCC3 para o slot 2
&HFCC4 para o slot 3
&H80 indica slot expandido
&H00 não expandido

SLTTBL – Indica que valor está saindo para o registro seletor de slot de expansão. Isto é válido somente quando o EXPTBL contém &H80, isto é, quando o slot é expandido.
Localização &HFFC5 – 4 bytes de comprimento.

SLTTBL &HFCC5 para o slot 0
&HFCC6 para o slot 1
&HFCC7 para o slot 2
&HFCC8 para o slot 3

Status de cada Página

SLTATR – Contém o que está em cada página para todas as páginas possíveis.
Localização &HFCC9 – 64 bytes de comprimento.

SLTATR &HFCC9 para o slot básico 0. O slot 0 de expansão página 0.
&HFCCA para o slot básico 0. O slot 0 de expansão página 1.
...
...
&HFD07 para o slot básico 3. O slot 3 de expansão página 2.
&HFD08 para o slot básico 3. O slot 3 de expansão página 3.

SLTWRK – área de trabalho para cada página. O uso depende do que existir na página. São usados dois bytes por página.

Localização &HFD09 – 128 bytes de comprimento; 2 bytes por página.

SLTWRK &HFD09 para o slot básico 0. O slot 0 de expansão página 0.
&HFD0A para o slot básico 0. O slot 0 de expansão página 0.
&HFD0B para o slot básico 0. O slot 0 de expansão página 1.
&HFD0C para o slot básico 0. O slot 0 de expansão página 1.
... ...
... ...
&HFD85 para o slot básico 3. O slot 3 de expansão página 2.
&HFD86 para o slot básico 3. O slot 3 de expansão página 2.
&HFD87 para o slot básico 3. O slot 3 de expansão página 3.
&HFD88 para o slot básico 3. O slot 3 de expansão página 3.

CAPÍTULO 51

PERIFÉRICOS

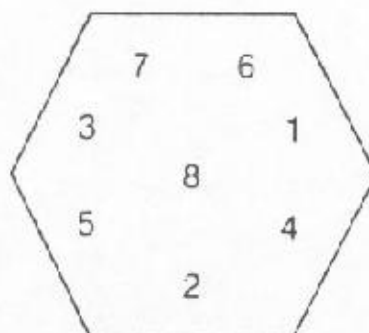
CASSETE

Os comandos seguintes estão associados com o uso de um cassete (descrições detalhadas são fornecidas no Livro de referência do BASIC).

CLOAD
CSAVE
MOTOR
SAVE
LOAD
BSAVE
BLOAD

Conector: conector DIN de 8 pinos, tipo 45326.

Tabela de sinais

Nº	SINAL	SENTIDO
1	GND	
2	GND	
3	GND	
4	CMTOUT	SAÍDA
5	CMTIN	ENTRADA
6	REM+	SAÍDA
7	REM–	SAÍDA
8	GND	

IMPRESSORA

Se o seu computador MSX estiver equipado com uma interface (conexão) de impressora, então você pode usar qualquer tipo de impressora que pertença ao padrão Centronics. Se o seu computador não tiver uma conexão de impressora embutida, então você deve comprar um cartucho de conexão de impressora que se conecte no slot de cartucho.

Algumas impressoras são equipadas com um conjunto de caracteres padrão do MSX. Essas impressoras compatíveis com o MSX são fabricadas por alguns fabricantes e dão a você a vantagem de ser capaz de imprimir todos os caracteres gráficos do MSX. Uma impressora não compatível com o MSX não será capaz de imprimir os caracteres gráficos; em vez disso, ela colocará um espaço para cada caractere gráfico.

Se você está usando uma impressora compatível com o MSX, então você deve dizer isso ao computador usando o comando SCREEN.

```
SCREEN,,,,0 seleciona uma impressora MSX
SCREEN,,,,1 seleciona uma impressora nao MSX
```

LPRINT
LPRINT USING

Para imprimir com uma impressora, você deve usar os comandos LPRINT e LPRINT USING, em vez de PRINT e PRINT USING. As suas funções são virtualmente as mesmas, exceto que LPRINT e LPRINT USING não irão fornecer qualquer exibição no modo de tela. Se você quiser que qualquer coisa seja impressa no modo de tela e na impressora ao mesmo tempo, você deve usar PRINT e LPRINT no seu programa.

LLIST lista o programa na impressora. Sua sintaxe é a mesma do comando LIST.

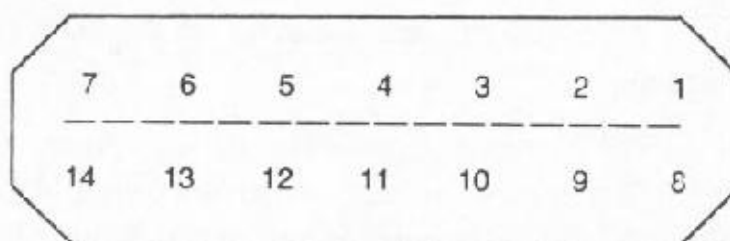
A função LPOS retorna a posição do cabeçote da impressora, e é análoga à função POS, que fornece a posição do cursor no modo de tela de texto.

Serão fornecidos maiores detalhes sobre impressoras quando você comprar a sua. Invariavelmente, o manual irá incluir programas de demonstração, escritos no BASIC da Microsoft que é compatível com o MSX-BASIC.

Conector: Tipo Centronics, 14 pinos Aphenol.

PINO Nº	NOME DO SINAL
1	PSTB
2	PDB0
3	PDB1
4	PDB2
5	PDB3
6	PDB4
7	PDB5
8	PDB6
9	PDB7
10	NC (sem função)
11	BUSY
12	NC (sem função)
13	NC (sem função)
14	GND

7 6 5 4 3 2 1

14 13 12 11 10 9 8

PORTA DO JOYSTICK

A maioria dos computadores MSX tem duas portas para joystick, enquanto outros tem somente uma; o BASIC pode suportar até duas. Tirando os joysticks, você pode conectar outros dispositivos como pads e paddles de jogos.

Conector: AMP de 9 pinos.

Joystick

Joystick são usados principalmente em jogos. Se o seu computador tiver duas portas para joystick, mas você tiver somente um, assegure-se de conectá-lo na porta JOYSTICK 1, pois a maioria dos programas comerciais assumem que quando você está usando somente um, ele está conectado nessa porta.

Para ler o status do joystick a partir do BASIC, use os comandos STICK e STRIG. STICK retorna em que direção está o joystick, e STRIG diz a você o status do botão de disparo do joystick.

O comando ON STRIG GOSUB fornece a você um interruptor de disparo. Veja a seção Manuseio de Eventos e Interrupções.

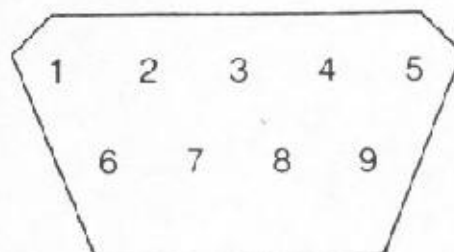
Paddle de Jogos

Você pode conectar até 12 paddles de jogos, 6 em cada porta de joystick. O status do paddle pode ser lido usando a função PDL (veja PDL).

Pad de Toque

Você pode conectar um pad de toque por porta de joystick.

PAD retorna o status do pad de toque (veja a seção de referência PAD no BASIC).

	SINAL	DIREÇÃO
1	FWD	ENTRADA
2	BACK	ENTRADA
3	LEFT	ENTRADA
4	RIGTH	ENTRADA
5	+5V	...
6	TRG 1	ENTRADA
7	TRG2	ENTRADA
8	SAÍDA	SAÍDA
9	GND	...

PORTA DO CARTUCHO

Lista dos Pinos de Sinal

ÍNDICE ANALÍTICO

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

OUTROS LIVROS NA ÁREA

Avalon Software – *O Livro Vermelho do MSX*

Burd/Moreira – *MSX – Jogos – volumes 1 a 3*

Burd/Moreira – *MSX – Comandos Básicos – Guia do Operador*

Burd – *Simulações no MSX*

Bussab – *MSX – Música*

Carvalho – *Assembler para o MSX*

Casari – *MSX com Disk Drive*

Hoffman – *MSX – Guia do Usuário*

Marriot – *MSX – O meu Primeiro Livro*

0-07-450423-1